Diplomarbeit

Ein Färbungsproblem in partitionierten Graphen

vorgelegt von
Christian Gießelbach

unter Anleitung von
Professor Dr. Rainer Schrader

Mathematisches Institut der Universität zu Köln
Sommersemester 2005

Inhaltsverzeichnis

Erklärung

Ich, Christian Gießelbach, erkläre hiermit, daß ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe angefertigt habe. Die benutzten Quellen und Hilfsmittel sind vollständig angegeben worden.

Köln, den 27. Juli 2005

Danksagung

Ich bedanke mich bei allen, die mich während des Studiums und dieser Diplomarbeit unterstützt haben.

Ein besonderer Dank gilt Hern Professor Dr. Schrader für die Themenstellung und Betreuung der Arbeit.

Für die Betreuung und das Lektorat der Arbeit danke ich außerdem Herrn Lars Kaderali.

Einleitung

Die DNA besteht aus zwei verbundenen, langen Ketten aus sogenannten Nukleotiden. Die einzelnen Nukleotide unterscheiden sich dadurch, welche der vier Basen Adenin (A), Guanin (G), Cytosin (C) und Thymin (T) an sie gekoppelt ist. Jedes Nukleotid des einen Stranges ist mit einem aus dem anderen verbunden, wobei A und T sowie C und G aneinander binden können. Durch diese Bindung ergibt sich ein Doppelstrang, in dem jeder der beiden einzelnen Stränge aus dem anderen rekonstruiert werden kann. Jeweils drei aufeinanderfolgende solcher Basenpaare kodieren ein Protein. Abbildung 1 zeigt ein Beispiel für einen solchen Doppelstrang. Wir beschränken uns in der Darstellung auf die Basen, da der Rest des Moleküls für unsere Zwecke unerheblich ist.

$$\begin{array}{cccccccccc} \rightarrow & A & C & G & A & C & \boxed{T} & T & A & \rightarrow \\ \leftarrow & T & G & C & T & G & \boxed{A} & A & T & \leftarrow \end{array}$$

Abbildung 1: Beispiel für einen DNA-Doppelstrang

Weite Teile des menschlichen Erbgutes, über 90%, sind immer gleich. Bei genetisch bedingten Krankheiten spielen die sogenannten "single nucleotide polymorphisms" (SNP) eine wichtige Rolle. Ein SNP ist ein abweichendes Basenpaar in diesen konstanten Bereichen des Erbgutes, wie etwa das in Abbildung 1 eingerahmte Paar. Beispielsweise wird Sichelzellenanämie durch einen solchen SNP verursacht. Die Analyse von SNPs ist folglich ein medizinisch wichtiges Gebiet. Dazu muß ein einzelnes Basenpaar aus einem DNA-Strang ausgelesen werden. Das heute gängige Verfahren dazu ist, den Doppelstrang in die beiden einzelnen Stränge zu zerlegen und durch Experimente herauszufinden, welche Base an der zu untersuchenden Stelle bindet. Dafür benötigt man einen sogenannten Primer, der neben der fraglichen Stelle an den Strang bindet (Abbildung 1). Zusammen mit Primern und markierten Basen kann in weiteren Arbeitsschritten ermittelt werden, welche Base an der Stelle auftritt, da sich die Base im DNA-Strang aus den Bindungsregeln ergibt.
Bei der Wahl der Primer gibt es zwei Faktoren, die variiert werden können. Zum einen kann der Primer unterschiedlich lang gewählt werden, zum anderen hat man die Wahl, aus welchem der beiden Einzelstränge die entsprechende Position ausgelesen wird. Typische Primer umfassen etwa 25 Nukleotide.
Um herauszufinden, ob eine Krankheit genetisch bedingt ist und gegebenenfalls die auslösenden Gene zu finden, müssen in einem DNA-Strang sehr viele solcher Ab-

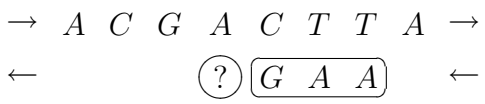


Abbildung 2: Ein Primer (unten eingerahmt) bindet neben der zu analysierenden Stelle an den DNA-Strang

fragen durchgeführt werden. Aus zeitlichen wie auch ökonomischen Gründen sollen möglichst viele der dafür notwendigen Abfragen gleichzeitig in einem Vorgang durchgeführt werden. Ein Problem stellt dabei die Wahl der Primer dar, denn es müssen einige Punkte von allen Primern erfüllt werden:

- die Primer dürfen nur an der jeweils auszulesenden Stelle binden
- sie dürfen nicht untereinander binden
- die unterschiedlichen Primer müssen in etwa bei gleichen Bedingungen wie Temperatur etc. binden.

Wenn Primer nicht miteinander verträglich sind, können diese zwei Abfragen nicht in demselben Vorgang erfolgen. Um möglichst viele Analysen in einem Experiment durchführen zu können, müssen die Primer folglich so gewählt werden, daß diese Konflikte vermieden werden und gleichzeitig nur eine geringe Anzahl unterschiedlicher Experimente notwendig wird.
In [6] wird ein graphentheoretisches Modell vorgestellt, mit dem sich das Problem der Primer-Auswahl lösen läßt. Dabei werden die möglichen Primer für ein Basenpaar als eine Gruppe von Knoten aufgefaßt, ein Konflikt wird durch die Verbindung dieser zwei Primer modelliert. Ziel ist es, aus jeder Gruppe einen Primer so auszuwählen, daß die gewählten Primer in möglichst wenige Gruppen eingeteilt werden können und daß zwischen den Primern einer Gruppe keine Konflikte auftreten.
Dieses Modell führt zu einer Verallgemeinerung eines der klassischen Probleme der Graphentheorie. Diese Verallgemeinerung ist über diese konkrete Anwendung hinaus interessant und lohnt daher auch eine isolierte Betrachtung. Im folgenden wird nach einer kurzen Einführung in die grundlegenden mathematischen Konzepte und das mathematische Modell in Kapitel 3 die generelle Schwierigkeit des Problems analysiert. In Kapitel 4 werden wir die bekannten und zwei neue Heuristiken vorstellen, und schließlich in Kapitel 5 einige damit erzielte Resultate vorstellen.

Kapitel 1

Grundlagen und Modellierung

1.1 Grundlagen der Graphentheorie

1.1.1 Begriffe und Notationen

Graphen

Ein *Graph* ist ein Tupel $G = (V, E)$ bestehend aus einer endlichen Menge V und $E \subset \{\{v, w\} \in V \,:\, v \neq w\}$. Die $v \in V$ heißen *Knoten*, die Elemente von E die *Kanten* des Graphen. Oft werden die Knoten als Punkte in der Ebene und Kanten als Linien zwischen den entsprechenden Punkten dargestellt.
Die Koten $w \in V$, für die $\{v, w\} \in E$ gilt, heißen die *Nachbarn* von v in G. Die Menge der Nachbarn von v wird mit $\delta(v)$ notiert. Zwei benachbarte Knoten werden auch *adjazent* genannt. Die Knoten v und w bezeichnet man als die *Endpunkte* der Kante $\{v, w\}$.
Der *Grad* eines Knotens v ist $\mathrm{d}(v) = |\delta(v)|$. Knoten mit Grad 0 werden *isoliert* genannt.

Teilgraphen

Oft interessiert man sich für Teilmengen eines Graphen. Dazu definieren wir die Menge aller Kanten, deren Endpunkte in einer gegebenen Knotenmenge liegen und umgekehrt:

$$E(V') = \{\{v, w\} \in E \,:\, v, w \in V'\} \quad \text{für } V' \subset V \tag{1.1}$$
$$V(E') = \{v \in V \,:\, \exists e \in E \text{ mit } v \in e\} \quad \text{für } E' \subset E \tag{1.2}$$

$E(V')$ heißt die von V' *induzierte Kantenmenge*, entsprechend $V(E')$ die von E' induzierte Knotenmenge.
Ein Teilgraph von $G = (V, E)$ ist ein Graph $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E(V')$, das heißt G' umfaßt eine Teilmenge der Knoten von G und allenfalls Kanten, die auch in G vorhanden sind. Meistens werden wir jedoch die Teilgraphen der Form $G' = (V', E(V'))$ oder $G' = (V(E'), E')$ betrachten, die sogannten *induzierten* Teilgraphen. Je nach Form spricht man auch von knoten- beziehungsweise kanteninduzierten Teilgraphen.

Besondere Knotenmengen

Wenn zwischen zwei Knoten v und w keine Kante verläuft, heißen die Knoten *unabhängig*. Eine Menge von Knoten heißt entsprechend unabhängig oder auch *stabil*, wenn keine Kante zwischen ihren Elementen verläuft.
Das Gegenteil einer stabilen Menge sind die *Cliquen*. Dies sind Teilgraphen, in denen alle Knoten zueinander adjazent sind. Wenn der gesamte Graph eine Clique ist, spricht man auch von einem *vollständigen Graphen*.
Ein Teilgraph $(\{v_1, \ldots, v_{l+1}\}, \{e_1, \ldots, e_l\})$ wird als *Pfad* von v_1 nach v_{l+1} bezeichnet, wenn $e_i = \{v_i, v_{i+1}\}$ für $1 \leq i \leq l$. Gilt zusätzlich $v_i \neq v_j$ für $i \neq j$, spricht man von einem *Weg*. Wege sind also Pfade, in denen jeder Knoten nur einmal auftritt. Wichtig ist, daß zwischen den Knoten eines Weges durchaus weitere Kanten verlaufen dürfen. Wenn zwischen je zwei Knoten eines Graphen ein Weg existiert, nennt man den Graphen *zusammenhängend*.
Eine besondere Form von Wegen sind die *Kreise*, bei denen Start- und Endpunkt des Pfades identisch sind. Wie schon bei Wegen können weitere Kanten zwischen den Knoten eines Kreises existieren. Oft wird auch nur die Knoten- oder Kantenmenge eines Kreises als Kreis bezeichnet.

Spezielle Graphen

Ein Graph wird als *Baum* bezeichnet, wenn er zusammenhängend und kreisfrei ist. Ist der Zusammenhang nicht gegeben, spricht man von einem *Wald*.

Definition 1.1 (bipartite und n-partite Graphen)**.** Ein Graph $G = (V, E)$ heißt *bipartit* wenn es eine Partition der Knotenmenge $V = V_1 \dot\cup V_2$ gibt (mit $V_1 \cap V_2 = \emptyset$), so daß oBdA gilt $\{v, w\} \in E \Leftrightarrow v \in V_1, w \in V_2$.
Allgemeiner heißt ein Graph n-partit, wenn die Knotenmenge so in $V_1, \ldots, V_n$ partitioniert werden kann, daß $\{v, w\} \in E$ mit $v \in V_i, w \in V_j$ nur dann möglich ist, wenn $i \neq j$.

Satz 1.2 (König, [8])**.** *Ein Graph ist genau dann bipartit, wenn er keinen Kreis ungerader Länge enthält. Es existiert ein Linearzeit-Algorithmus, der zu einem gegebenen Graphen entweder eine Bipartition oder einen ungeraden Kreis findet.*

1.1.2 Graphenfärbungen

Das Graphenfärbungsproblem ist eines der klassischen Probleme der Graphentheorie. Die Aufgabe dabei ist, die Knoten so in disjunkte Gruppen einzuteilen, daß zwischen den Knoten einer Gruppe keine Kante verläuft. Aus der Anschauung motiviert spricht man davon, die Knoten zu färben, wobei benachbarte Knoten unterschiedliche Farben erhalten müssen. Formal ist eine Färbung eine Abbildung $c : V \to 1, \ldots, k$ mit $\{v, w\} \in E \Rightarrow c(v) \neq c(w)$. Wir gehen immer davon aus, daß c surjektiv ist, also alle Werte der Bildmenge auch angenommen werden. Eine

k-Färbung ist eine Färbung, die k Farben benutzt. Es ist durchaus möglich, daß ein Graph mehrere unterschiedliche k-Färbungen hat. Das Graphenfärbungsproblem ist, zu einem gegebenen Graphen das kleinste k zu bestimmen, für das eine k-Färbung des Graphen existiert.

Definition 1.3 ($\chi(G)$)**.** Sei $G = (V, E)$ ein Graph. Die *chromatische Zahl* $\chi(G)$ ist die kleinste Anzahl Farben, mit der G gefärbt werden kann:

$$\chi(G) = \min \{k \in \mathbb{N} \ : \ \text{es existiert eine } k\text{-Färbung von } G\} \tag{1.3}$$

Das Graphenfärbungsproblem ist also das Problem, $\chi(G)$ zu bestimmen.

Bemerkung. Im Kontext der Graphenfärbung sind die bipartiten Graphen genau die Graphen, die 2-färbbar sind. Die chromatische Zahl eines n-partiten Graphen ist höchstens n. Da die Kanten nur zwischen Knoten aus verschiedenen Partitionen verlaufen, ist es eine zulässige Färbung, für jede Partition eine Farbe einzuführen und diese allen Knoten der Partition zuzuweisen.

Knoten mit derselben Farbe sind per Definition paarweise unabhängig und bilden eine stabile Menge. Diese Mengen werden die *Farbklassen* der Färbung genannt.

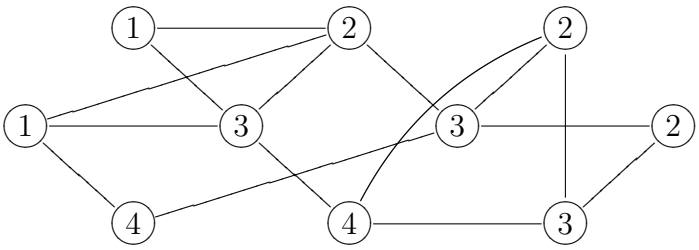
Graphenfärbungen werden bei vielen Optimierungsproblemen zur Modellierung benutzt, insbesondere bei Routing- und Schedulingproblemen. Allerdings ist das Graphenfärbungsproblem selbst bereits schwierig:

Satz 1.4 (R. Karp, [7])**.** *Das Graphenfärbungsproblem ist NP-vollständig.*

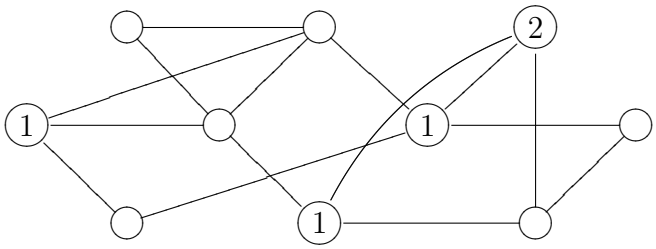
1.2 Modellierung des Primer-Designs

Das Problem des Primer-Designs hat zwei Kernbestandteile. Zum einen müssen die Primer gewählt werden, zum anderen müssen die gewählten Elemente in Gruppen eingeteilt werden, die jeweils nur miteinander verträgliche Primer enthalten. In [6] beschreiben Kaderali, Deshpande, Nolan und White ein graphentheoretisches Modell für dieses Problem, das eine Verallgemeinerung des klassischen Graphenfärbungsproblems darstellt.
Das Graphenfärbungsproblem eignet sich sehr gut, um die Gruppeneinteilung zu modellieren. Dazu muß ein Graph konstruiert werden, der die Problemstruktur widerspiegelt. Eine Möglichkeit dazu ist, jeden Knoten des Graphen mit einem gewählten Primer zu identifizieren. Zwei Knoten werden genau dann durch eine Kante verbunden, wenn die entsprechenden Primer nicht miteinander verträglich sind, also nicht beide gemeinsam in einem Experiment eingesetzt werden können.
Es ist leicht zu sehen, daß jede zulässige Gruppeneinteilung der Primer einer Färbung des Graphen entspricht, wobei jede Gruppe genau die Knoten einer Farbe umfaßt.

Diese Äquivalenz folgt unmittelbar aus der Tatsache, daß die entsprechenden Knoten nur dann dieselbe Farbe haben können, wenn sie keine Kante verbindet. Damit können nach Konstruktion Knoten einer Farbe nie zu unverträglichen Primern gehören. Andererseits kann eine Gruppe paarweise verträglicher Primer stets dieselbe Farbe erhalten. Somit entspricht die kleinste Anzahl notwendiger Experimente genau der chromatischen Zahl des auf diese Art konstruierten Graphen.
Zusätzlich zur Gruppeneinteilung besteht das Problem, für jedes zu analysierende Nukleotid im DNA-Strang einen Primer aus einer ganzen Gruppe von Kandidaten zu wählen. Dabei soll die Wahl natürlich so erfolgen, daß die gewählten Primer in möglichst wenige Gruppen eingeteilt werden können. Im Bezug auf das gerade beschriebene Färbungsproblem heißt dies, daß die chromatische Zahl des von den ausgewählten Primern induzierten Graphen so klein wie möglich sein soll. Die Kombination der beiden Probleme führt zu einer Variante des Färbungsproblems. Der Graph wird wie vorhin so konstruiert, daß die Knoten den Primern und die Kanten den Konflikten entsprechen. Der einzige Unterschied ist, daß der Graph nun statt der gewählten Primer alle möglichen Kandidaten enthält. In diesem Graphen muß bei der Gruppeneinteilung nicht jedem Knoten eine Farbe zugewiesen werden, schließlich wird nur je ein Primer für jedes auszulesende Nukleotid benötigt. Man hat bei der Färbung also die Wahl zwischen mehreren Knoten, die alle zu Primern für dieselbe Stelle im DNA-Strang gehören, aus denen nur jeweils einer ausgesucht werden muß. Wichtig ist nur, daß zu jedem Nukleotid auch ein Primer gewählt wird.
Damit entsteht eine Variante des Graphenfärbungsproblems, in dem die Knotenmenge in mehrere Gruppen zerfällt, hier eben die Primer für eine bestimmte Stelle. Die Aufgabe ist, aus jeder dieser Gruppen je einen Knoten so zu wählen, daß diese ausgesuchten Knoten mit möglichst wenigen Farben gefärbt werden können. Diese Variante des Graphenfärbungsproblem wird als *Partition Coloring Problem* bezeichnet.



(a) Ein partitionierter Graph



(b) Eine Färbung des Graphen

Abbildung 1.1: Beispiel für das verallgemeinerte Färbungsproblem. 1.1(a): Alle Knoten mit derselben Nummer bilden eine Gruppe, aus der jeweils ein Knoten gewählt werden muß. 1.1(b): Eine optimale Auswahl und deren Färbung

Kapitel 2

Verwandte Probleme

Das Partition Coloring Problem ist ein Beispiel für die Verallgemeinerung graphentheoretischer Fragestellungen auf partitionierte Graphen. In vielen Fällen kann man den Lösungsbegriff so anpassen, daß in einer Optimallösung mindestens ein Knoten aus jedem Block auftreten muß.

2.1 Verallgemeinerte aufspannende Bäume

Die Frage nach einem minimal gewichteten aufspannenden Baum ist eine der ältesten Fragestellungen in der Graphentheorie. Wenn ein ungerichteter Graph $G = (V, E)$ und eine Gewichtsfunktion $c : E \to \mathbb{R}_+$ gegeben sind, soll ein kreisfreier, zusammenhängender Teilgraph von G gefunden werden, der minimales Gewicht hat. Dieses Problem ist mit Greedyalgorithmen effizient lösbar, siehe z.B. [9], Kapitel 6.
Die Verallgemeinerung dieses Problems auf partitionierte Graphen ist in den letzten Jahren ausführlich erforscht worden, da diese beim Design von VLSI-Chips und Netzwerken auftritt. Die Forderung nach einem aufspannenden Baum wird wie oben beschrieben so abgeschwächt, daß der optimale Baum nur mindestens einen Knoten aus jeder Partition enthalten muß. Dieses verallgemeinerte Problem wird als generalized minimum spanning tree oder auch group Steiner tree problem bezeichnet, wobei die erste Formulierung benutzt wird um anzuzeigen, daß genau ein Knoten aus jedem Block gewählt werden soll.
Die erste Formulierung des allgemeinen Problems stammt von Reich und Widmayer [13], die auch bewiesen haben, daß das verallgemeinerte Problem im Gegensatz zur Berechnung der üblichen minimalen aufspannenden Bäume NP-vollständig ist. Im Jahr 2000 haben Garg, Konjevod und Ravi ein Verfahren mit polylogarithmischer Approximationsgüte für das group Steiner tree problem vorgestellt [4].
Das group Steiner tree Problem kann selbst weiter verallgemeinert werden. So haben z.B. Konjevod und Ravi auch das Covering Steiner Problem untersucht, bei dem aus jeder Partition eine bestimmte Anzahl Knoten gewählt werden muß. Penn und Rozenfeld [12] stellten 2003 einen Algorithmus zur Approximation einer Variante vor, bei der eine bestimmte Menge Pfade zwischen den Repräsentanten existieren soll.

2.2 Partitionierte Matchings

Ein Matching in einem Graphen $G = (V, E)$ ist eine Teilmenge von Kanten, so daß keine zwei Kanten einen Knoten gemeinsam haben. Das Matchingproblem ist ein weiteres klassisches Problem in der Graphentheorie. Es ist sehr genau erforscht, siehe wiederum [9]. Die bekanntesten Varianten sind das Kardinalitätsmatching, bei dem nach einem Matching mit möglichst vielen Kanten gesucht wird, und das gewichtete Matchingproblem. Alle diese Varianten des Problems sind effizient lösbar. Auch bei diesem Problem ist die Verallgemeinerung auf partitionierte Graphen offensichtlich. Wie auch beim Graphenfärbungsproblem, das in den folgenden Kapiteln behandelt werden wird, steigt die Komplexität des Problems durch die Verallgemeinerung.

Kapitel 3

Das Partion Coloring Problem (PCP)

3.1 Problembeschreibung

Wir betrachten wie in Kapitel 1 dargestellt ein Graphenfärbungsproblem, bei dem nur je ein Knoten aus einer ganzen Gruppe gefärbt werden muß. In diesem Abschnitt werden wir einige fundamentale Eigenschaften des Problems vorstellen und anschließend einige Resultate zur Komplexität des gesamten Problems sowie der Berechnung unterer Schranken vorstellen. Zunächst benötigen wir jedoch einige Begriffe.

Definition 3.1 (partionierter Graph)**.** Sei $G = (V, E)$ ein Graph. Wir nennen den Graphen *partitioniert*, wenn eine Einteilung der Knotenmenge V in k disjunkte Mengen gegeben ist, also

$$V = \bigcup_{i=1}^{k} V_i \text{ und } V_i \cap V_j = \emptyset \text{ wenn } i \neq j$$

Die einzelnen Partitionen V_i nennen wir *Blöcke*, die Anzahl k der Blöcke die *Blockzahl* des Graphen.

Definition 3.2 (Repräsentantensytem)**.** Sei $G = (V = V_1 \dot\cup \cdots \dot\cup V_k, E)$ ein partitionierter Graph. Ein *Repräsentantensystem* für G ist eine Knotenmenge $R \subset V$, die aus jeder der Partitionen V_i genau einen Knoten enthält, also $|R \cap V_i| = 1$ für alle i im Bereich $1, \ldots, k$. Jeder Knoten in R heißt der *Repräsentant* seines jeweiligen Blockes.

Definition 3.3 (Partition Coloring Problem)**.** Sei $G = (V = V_1 \dot\cup \cdots \dot\cup V_k, E)$ ein partitionierter Graph. Wir betrachten das Problem, ein Repräsentantensystem mit minimaler chromatischer Zahl zu finden und eine optimale Färbung anzugeben. Dieses Problem wird als Partition Coloring Problem (kurz: PCP) bezeichnet. Die kleinste chromatische Zahl eines Repräsentantentsystems bezeichnen wir mit $\chi_P(G)$, das heißt

$$\chi_P(G) = \min_R \chi(R, E(R)) \text{ mit } |R \cap V_i| = 1 \text{ für alle } i \tag{3.1}$$

Das Partition Coloring Problem ist somit, χ_P zu bestimmen und ein Repräsentantensystem zu finden, für das das Minimum in (3.1) angenommen wird.

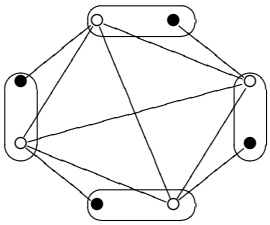


Abbildung 3.1: Unterschied zwischen χ_P-Färbung und χ-Färbung eines Graphen

3.2 Unterschiede zwischen χ und χ_P

Es ist klar, daß in jedem Graphen $\chi_P(G) \leq \chi(G)$ gilt. Wenn eine optimale Färbung des gesamten Graphen G gegeben ist, ist jeder Knoten zulässig mit einer Farbe aus dem Bereich $1, \ldots, \chi(G)$ gefärbt. Folglich ist auch jeder Teilgraph zulässig gefärbt, und kann auf keinen Fall mehr Farben erfordern.
Ein einfacher Graph, in dem $\chi_P < \chi$ gilt, wird in Abbildung 3.1 gezeigt. Zwischen den vier Knoten "•" verläuft keine Kante, und da sie alle Blöcke überdecken, bilden sie auch ein Repräsentantensystem für den Graphen. Somit gilt $\chi_P(G) = 1$. Andererseits bilden die Knoten "○" eine Clique. Für diese Knoten sind also bereits vier Farben notwendig. Man sieht leicht, daß vier Farben auch für den gesamten Graphen ausreichen, und folglich gilt $\chi(G) = 4$.
An diesem Graph kann man auch sehen, daß nicht unbedingt eine χ-Färbung des ganzen Graphen existiert, in der die Knoten eines optimalen Repräsentantensystems die Farben aus einer χ_P-Färbung haben. Wenn die vier Knoten mit Grad 1 mit einer einzigen Farbe gefärbt werden, werden für den gesamten Graphen 5 Farben benötigt. Da jeder der vier Knoten "○" zu einem der Knoten "•" adjazent ist, ist die erste Farbe für keinen der Knoten aus der Clique zulässig. Da diese wie bereits beschrieben unterschiedliche Farben erhalten müssen, werden 4 zusätzliche Farben benötigt.

Die Konstruktion von Abbildung 3.1 kann leicht so erweitert werden, daß die Differenz zwischen $\chi(G)$ und $\chi_P(G)$ beliebig groß wird. Wir werden im Beweis des entsprechenden Satzes allerdings eine andere Konstruktion verwenden, so daß die erzeugten Graphen eine zusätzliche Eigenschaft haben. Es soll verhindert werden, daß die Nachbarmenge eines Knotens eine Obermenge der Nachbarn eines anderen Knotens aus dem selben Block ist. In Abschnitt 4.2 werden wir zeigen, daß in diesem Fall der Knoten mit der größeren Nachbarmenge bei der Wahl eines Repräsentantensystems nicht betrachtet werden muß und aus dem Graphen entfernt werden kann. Genau diese Situation tritt bei dem Graphen aus Abbildung 3.1 ein. Er würde so auf die vier Knoten "•" reduziert.

Satz 3.4. *Es gibt eine Folge von Graphen $(G_n)_{n\in\mathbb{N}}$, so daß $\chi(G_k) = k$ und $\chi_P(G_k) = 1$ für alle $k \in \mathbb{N}$ mit $k \geq 2$.*

Beweis. Die folgenden Graphen erfüllen den Satz:

$$\begin{aligned} & G_k = (V_k = V^{(1)} \dot\cup \cdots \dot\cup V^{(k)}, E_k) \\ \textit{mit} \qquad & V^{(i)} = \{v_1^{(i)}, v_2^{(i)}, v_3^{(i)}\} \quad 1 \le i \le k \\ & E_k = \left\{ \{v_1^{(p)}, v_1^{(q)}\}, \{v_3^{(p)}, v_3^{(q)}\}, \{v_2^{(p)}, v_3^{(q)}\}, \{v_2^{(p)}, v_1^{(q)}\} \ : \ p \neq q \right\} \end{aligned} \tag{3.2}$$

Offensichtlich gilt $\chi(G_k) \geq k$, da die $v_1^{(i)}$ und $v_3^{(i)}$ jeweils eine Clique der Größe k bilden. Da innerhalb jedes Blockes $V^{(i)}$ keine Kanten verlaufen, ist der Graph k-partit und hat somit eine k-Färbung. Eine mögliche k-Färbung ergibt sich, wenn alle Knoten eines Blockes dieselbe Farbe erhalten.
Zwischen zwei Knoten $v_2^{(p)}$ und $v_2^{(q)}$ verläuft wieder keine Kante, so daß $\chi_P(G_k) = 1$. Mit dem gleichen Argument wie eben ergibt sich auch für diese Graphen, daß keine k-Färbung des gesamten Graphen möglich ist, bei der alle $v_2^{(p)}$ dieselbe Farbe erhalten. Denn jeder der Knoten $v_3^{(q)}$ ist zu $k-1$ dieser Knoten adjazent, so daß er nicht diese Farbe erhalten kann. Da die $v_3^{(q)}$ eine Clique bilden, werden in diesem Fall mindestens $k+1$ Farben benötigt.
Der Graph G_3 wird in Abbildung 3.2 gezeigt. □

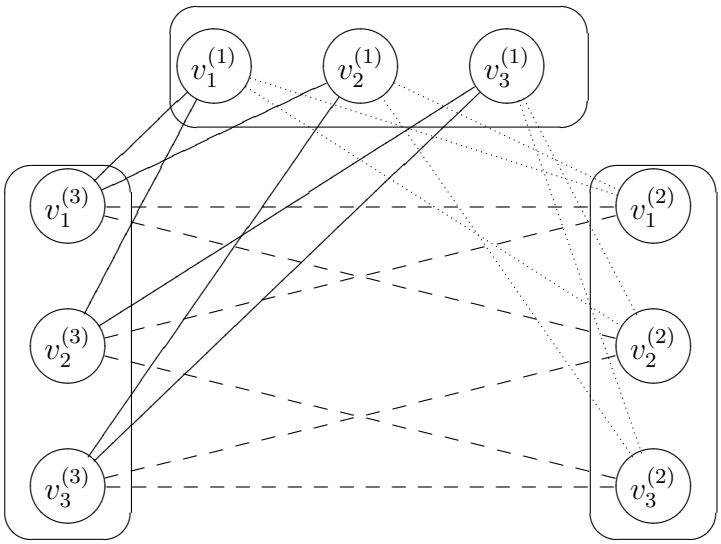


Abbildung 3.2: G_3 aus Satz 3.4

3.3 Komplexität des allgemeinen PCP

Das klassische Graphenfärbungsproblem kann auch als eine spezielle Form des Partition Coloring Problems aufgefaßt werden. Dazu betrachtet man jeden Knoten als eigenen Block, das heißt $V = \{v_1\}\dot\cup \cdots \dot\cup \{v_n\}$. Bereits diese Teilmenge von Instanzen von PCP ist NP-vollständig (R. Karp, [7]), so daß das allgemeine Problem mindestens genau so schwer zu lösen ist.

Satz 3.5. *Das Partition Coloring Problem ist NP-vollständig.*

Beweis. Zum Beweis der NP-Vollständigkeit eines Problems sind zwei Eigenschaften zu zeigen:

- Das Problem muß in NP enthalten sein. Dazu muß gezeigt werden, daß für einen Lösungskandidaten effizient entschieden werden kann ob er zulässig ist. In diesem Fall bedeutet dies, daß für jede gefärbte Knotenmenge schnell geprüft werden kann, ob sie aus jedem Block einen Knoten enthält und ob die Färbung zulässig ist.

- Das Problem muß NP-schwer sein, also ein schwerstes Problem für NP. Dafür ist zu zeigen, daß alle Instanzen der Probleme in NP effizient auf Instanzen dieses Problems abgebildet werden können, so daß sich die Optimallösungen entsprechen. Da zwei effiziente Abbildungen hintereinander ausgeführt wieder eine effiziente Abbildung ergeben, langt es zu zeigen, daß die Instanzen eines NP-vollständigen Problems entsprechend auf dieses Problem transformiert werden können.

Sei $G = (V_1\dot\cup \cdots \dot\cup V_k, E)$ ein beliebiger Graph. $PCP \in NP$, denn zu jeder Knotenmenge $V' \subset V$ kann offensichtlich in polynomieller Zeit entschieden werden, ob aus jeder Partition mindestens ein Knoten gewählt worden ist und ob V' zulässig gefärbt ist.
Die Behauptung folgt nun unmittelbar aus der Tatsache, daß, wie oben erwähnt, jede Instanz des Graphenfärbungsproblems auch eine PCP-Instanz ist, wenn man die triviale Partitionierung von V mit jedem Knoten als Block betrachtet. Da nach Satz 1.4 das Graphenfärbungsproblem NP-vollständig ist, folgt die Behauptung. □

In diesem Beweis nutzt man aus, daß man eine Verallgemeinerung eines schwierigen Problems betrachtet. Die Frage ist, ob Instanzen mit größeren Blöcken eventuell einfacher zu behandeln sind. Dies ist jedoch nicht der Fall, wie die folgende Variante des Satzes zeigt:

Satz 3.6 (Li und Simha [10])**.** *PCP ist NP-vollständig, auch wenn die minimale Anzahl Knoten in einem Block mindestens $c \in \mathbb{N}, c > 1$ ist.*

Beweis. Sei $c \in \mathbb{N}$ beliebig, aber fest gewählt und $G = (V, E)$ ein beliebiger Graph. Wir zeigen, daß sich das Graphenfärbungsproblem auch auf eine PCP-Instanz in einem Graphen $G' = (V', E')$ abbilden läßt, in der jeder Block mindestens c Knoten enthält.
Den entsprechenden Graphen G' konstruiert man, indem jeder Knoten aus V in V' durch c Knoten ersetzt wird, wobei jeder dieser Knoten dieselben Nachbarn wie der ursprüngliche Knoten hat. In G' besteht jeder Block aus c Knoten, doch jedes Repräsentantensystem ist isomorph zu G, so daß $\chi_P(G') = \chi(G)$ ist. Abbildung 3.3 zeigt noch einmal genau die Konstruktion von G'. □

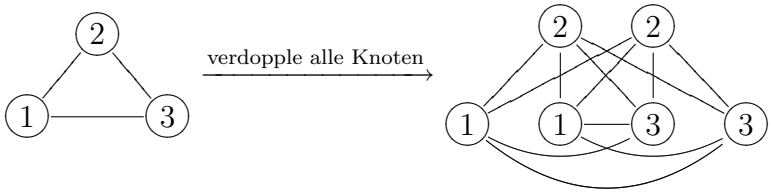


Abbildung 3.3: Beispiel zur Konstruktion von G' in Satz 3.6 mit $c = 2$

3.4 Stabile Repräsentantensysteme und untere Schranken

Durch jede zulässige Färbung eines Repräsentantensystems wird eine obere Schranke für die Farbzahl einer Optimallösung bestimmt, da auf keinen Fall mehr Farben benutzt werden müssen. Um jedoch die Qualität einer Lösung abschätzen zu können, muß man wissen, wie groß der Abstand zum Optimum höchstens ist. Dazu müssen untere Schranken für χ_P bekannt sein, die man statt des Optimums zur Bewertung heranzieht. Diese Schranken sollten selbstverständlich "möglichst groß" sein, um eine gute Näherung an die Optimallösung χ_P zu liefern. Offensichtlich wird für jedes Repräsentantensystem mindestens eine Farbe benötigt, so daß 1 die triviale untere Schranke ist.
In diesem Abschnitt werden wir zeigen, daß im Gegensatz zum klassischen Färbungsproblem nicht effizient entschieden werden kann, ob es eine Lösung gibt, die mit einer einzigen Farbe auskommt. Ein Graph hat genau dann eine 1-Färbung, wenn es keine einzige Kante gibt. In dem hier betrachteten Problem muß dagegen geprüft werden, ob es eine stabile Menge gibt, die aus jedem Block mindestens einen Knoten enthält. Wir werden zeigen, daß bereits dieses Problem NP-vollständig ist und ableiten, daß die Berechnung unterer Schranken generell schwierig ist.
Im nächsten Abschnitt werden wir das Reduktionsproblem XSAT vorstellen, auf das diese Frage zurückgeführt werden kann. Danach werden wir die entsprechenden Resultate für das Färbungsproblem vorstellen.

3.4.1 Das Exact-SAT-Problem (XSAT)

Das SAT-Problem (von Satisfiability) ist das zentrale Problem der Komplexitätstheorie. Es ist das erste Problem, das als NP-vollständig erkannt wurde. Dabei werden Formeln aus Variablen betrachtet, die nur die Werte 0 oder 1 annehmen dürfen, sogenannte *Booleschen Variablen*. Für diese Variablen werden die folgenden Operationen definiert:

Definition 3.7 (Disjunktion und Konjunktion)**.** Seien x und y zwei Boolesche Variablen. Man schreibt:

$$\begin{aligned} &\text{Disjunktion:} && x \vee y = 1 \;\Leftrightarrow\; (x,y) \neq (0,0) && (3.3)\\ &\text{Konjunktion:} && x \wedge y = 1 \;\Leftrightarrow\; x = y = 1 && (3.4)\\ &\text{Negation:} && \overline{x} = 1 \;\Leftrightarrow\; x = 0 && (3.5) \end{aligned}$$

Die Disjunktion wird auch "logisches Oder", die Konjunktion "logisches Und" genannt.

Für das SAT-Problem wird ein bestimmter Typ Ausdruck untersucht, der aus diesen drei Operationen zusammengesetzt wird, die SAT-Formeln.

Definition 3.8 (SAT-Formel)**.** Sei $X = \{x_1, \ldots, x_k\}$ eine endliche Menge von $\{0,1\}$-Variablen. Ein *Literal* ist eine einzelne Variable oder deren Negation. Eine *Klausel* ist die Disjunktion mehrer Literale, also ein Ausdruck der Form

$$x_1 \vee \overline{x_2} \vee x_3. \tag{3.6}$$

Eine *SAT-Formel* ist eine Konjunktion von derartigen Klauseln, z.B.

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \tag{3.7}$$

Eine konkrete Zuweisung der Werte 0 oder 1 zu den einzelnen Variablen einer Formel nennen wir eine *Belegung* der Variablen.

Definition 3.9 (SAT-Problem)**.** Gegeben sei eine SAT Formel. Das SAT-Problem ist zu entscheiden, ob es eine Belegung gibt, so daß das Ergebnis der Formel 1 ist. Wenn es eine solche Belegung gibt, heißt die Formel *erfüllbar*. Beispielsweise erfüllt die Belegung $(x_1, x_2, x_3, x_4) = (1, 0, 0, 1)$ die Formel (3.7).

Es ist nach der Rechenregel (3.4) klar, daß eine Belegung, die die gesamte Formel erfüllt, auch jede einzelne Klausel erfüllen muß.

Definition 3.10 (k-SAT)**.** Das k-SAT Problem ist das SAT-Problem, bei dem in jeder Klausel exakt k Literale auftreten.

Satz 3.11 (Cook [2])**.** *SAT ist NP-vollständig. Für $k \geq 3$ ist auch k-SAT NP-vollständig.*

Um zu zeigen, daß stabile Repräsentantensysteme schwer zu finden sind, betrachten wir eine Verschärfung der oben eingeführten Erfüllbarkeit von Klauseln und Formeln, die *exakte Erfüllbarkeit*.

Definition 3.12 (exakte Erfüllbarkeit)**.** Eine Belegung der Variablen erfüllt eine Klausel *exakt*, wenn genau ein Literal den Wert 1 hat und alle anderen 0. Entsprechend wird eine Formel exakt erfüllt, wenn dies für alle Klauseln zutrifft.

Es ist offensichtlich, daß jede exakt erfüllte Formel auch erfüllt ist. Es gibt jedoch Formeln, die zwar erfüllbar, aber nicht exakt erfüllbar sind. Dazu ein Beispiel:

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \tag{3.8}$$

Diese Formel wird z.B. durch $(x_1, x_2, x_3, x_4) = (0, 1, 0, 0)$ erfüllt, doch es gibt keine exakt erfüllende Belegung. Dies sieht man schnell, wenn man versucht, die erste Klausel exakt zu erfüllen. Jede der drei Möglichkeiten dies zu tun führt zwingend zu Belegungen, in denen eine der beiden anderen Klauseln entweder nicht oder durch mehr als ein Literal erfüllt ist.

Definition 3.13 (Exact-SAT-Problem)**.** Gegeben sei eine SAT-Formel. Das Exact-SAT-Problem ist zu entscheiden, ob diese Formel exakt erfüllbar ist.
Diese Variante des SAT-Problems bezeichnen wir kurz auch als XSAT. Analog zu Definition 3.10 ist das k-XSAT-Problem das XSAT-Problem, bei dem in jeder Klausel genau k Literale vorkommen.

Im Gegensatz zur normalen Erfüllbarkeit ist XSAT schon schwer zu entscheiden, wenn keine Negationen in der Formel auftreten. Wenn in einer Klausel mehrere Literale den Wert 1 haben dürfen, kann man in diesem Fall einfach alle Variablen auf 1 setzen und so die Formel erfüllen. Bei XSAT ist bereits dieses Problem NP-vollständig. Dies ist eine Konsequenz aus dem folgenden Satz.

Satz 3.14 (T. J. Schaefer [14])**.** *Seien $S_1, \ldots, S_m$ Teilmengen einer endlichen Grundmenge S mit $|S_i| \leq 3$. Das Problem, festzustellen, ob eine Menge $T \subsetneqq S$ mit $\forall i : |T \cap S_i| = 1$ existiert, ist NP-vollständig.*

Korollar 3.15. *3-XSAT ist NP-vollständig, selbst wenn keine Negationen auftreten.*

Beweis. Die in Satz 3.14 betrachtete Fragestellung ist äquivalent zum 3-XSAT Problem. Dazu führen wir für jedes Element der Grundmenge S eine Variable ein. Die Klauseln sollen den Teilmengen S_i entsprechen und enthalten genau die Variablen zu den entsprechenden Elementen. Einer Teilmenge $\{s_1, s_2, s_3\}$ entspricht so die Klausel $(x_1 \vee x_2 \vee x_3)$. Wenn es eine exakt erfüllende Belegung gibt, erhält darin genau eine Variable aus jeder Klausel den Wert 1. Diese Variablen entsprechen zusammen der Menge T aus dem Satz von Schaefer (3.14). □

3.4.2 NP-Vollständigkeit der Berechnung unterer Schranken

Eine wichtige Teilfrage bei der Analyse von PCP-Instanzen ist die Frage nach einem stabilen Repräsentantensystem, da ein solches System zwangsläufig eine Optimallösung ist. Leider führt die Erkenntnis, daß es ein solches System nicht gibt, nicht zu guten unteren Schranken. Die einzige Garantie ist, daß mindestens zwei Farben nötig sind. Damit ist keine sinnvolle Aussage über die Qualität eines Repräsentantensystems mit vielen Farben möglich. Dazu wären bessere Informationen notwendig, insbesondere darüber, wie groß der Abstand zwischen der unteren Schranke und χ_P ist.
In diesem Abschnitt werden wir allerdings zeigen, daß bereits die vermeintlich einfachere Frage nach stabilen Repräsentantensystemen NP-vollständig ist. Außerdem wird dieses Problem durch jede untere Schranke außer der trivialen Untergrenze 1 implizit gelöst, so daß eine effiziente Berechnung guter unterer Schranken nicht zu erwarten ist.

Satz 3.16. *In allgemeinen partitionierten Graphen ist es ein NP-vollständiges Problem festzustellen, ob ein stabiles Repräsentantensystem existiert.*

Beweis. Wir zeigen die Reduktion des 3-XSAT-Problems aus dem vorangegangenen Abschnitt auf dieses Problem. Sei f eine SAT-Formel, wobei keine Negationen auftreten und jede Klausel nicht mehr als drei Literale enthält. Seien f_i, $1 \leq i \leq k$, die Klauseln von f, also

$$f(x_1, \ldots, x_n) = \bigwedge_{i=1}^{k} f_i(x_1, \ldots, x_n) \tag{3.9}$$

Zum Beweis des Satzes müssen wir einen partitionierten Graphen konstruieren, in dem genau dann ein unabhängiges Repräsentantensystem existiert, wenn die Formel f exakt erfüllbar ist. Dazu führen wir für jedes Literal jeder Klausel einen Knoten ein:

$$\begin{aligned} V_i &:= \left\{x_j^i \ : \ x_j \in f_i, 1 \leq j \leq n\right\} \quad \text{für } 1 \leq i \leq k \\ V &= V_1 \dot\cup \cdots \dot\cup V_k \end{aligned} \tag{3.10}$$

Die Knotenmenge V_i entspricht mit dieser Definition den Variablen, die in der Klausel f_i vorkommen. Zu beachten ist, daß der Knoten x_j^i nur dann im Graphen enthalten ist, wenn die Variable x_j in Klausel f_i vorkommt. Ansonsten sagen wir, der Knoten x_j^i existiere nicht.
Die Kanten wählen wir wie folgt:

$$\{x_i^p, x_j^q\} \in E \ :\Leftrightarrow \ x_i^q \text{ oder } x_j^p \text{ existiert, } p \neq q \text{ und } i \neq j \tag{3.11}$$

Eine Kante verbindet also genau dann zwei Knoten, wenn die Variable zu einem der beiden Knoten auch in der Klausel vorkommt, zu der der andere gehört.
Abbildung 3.4 zeigt einen Graphen, der auf diese Art aus einer Formel konstruiert worden ist.

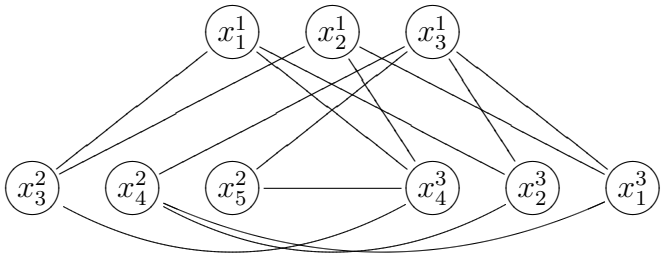


Abbildung 3.4: Beispiel für die Konstruktion eines Graphen aus der SAT-Formel $f(x_1, \ldots, x_5) = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2 \vee x_4)$

Zu zeigen ist nun, daß in dem so konstruierten Graphen tatsächlich die stabilen Repräsentantensysteme genau den exakt erfüllenden Belegungen der Formel f entsprechen.
Sei $x_1, \ldots, x_n$ eine Belegung der Variablen, die f exakt erfüllt. Für $x_i = x_j = 1$ gilt dann für alle p, q (sofern die entsprechenden Knoten existieren):

$$\{x_i^p, x_j^q\} \notin E$$

Im Fall $p = q$ können nicht beide Knoten x_i^p und x_j^p existieren, weil jede Klausel von f exakt erfüllt ist. Wenn $p \neq q$ gilt, existiert die Kante $\{x_i^p, x_j^q\}$ genau dann, wenn oBdA auch x_j^p existiert, im Widerspruch dazu daß f exakt erfüllt wird. Folglich sind die Knoten, die Variablen mit Wert 1 entsprechen, unabhängig. Da jede Klausel erfüllt wird, wird so auch jeder Block überdeckt.
Sei nun umgekehrt ein unabhängiges Repräsentantensystem R von G gegeben. Gezeigt wird, daß die Repräsentanten genau den 1-Variablen einer exakt erfüllenden Belegung entsprechen. Damit auf diese Weise eine Variablenbelegung induziert wird, müssen entweder alle Knoten zu einer Variablen in R enthalten sein oder keiner dieser Knoten. Angenommen es existiert ein unabhängiges Repräsentantensystem, das diese Forderung nicht erfüllt, d.h. oBdA $x_1^1 \in R$, aber $x_1^2 \notin R$. Sei x_j^2 der Repräsentant von V_2. Dann gilt nach (3.11) $\{x_1^1, x_j^2\} \in E$, da x_1^2 ja existiert. Dies ist allerdings ein Widerspruch zur Unabhängigkeit von R.
Es bleibt noch zu zeigen, daß f exakt erfüllt ist. Angenommen, in Klausel f_p seien mehrere Literale erfüllt, etwa durch x_i und x_j (mit $i \neq j$). Da aus jedem Block nur je genau ein Knoten gewählt worden ist, müssen diese von zwei Knoten x_i^p und x_j^q ($p \neq q$) aus unterschiedlichen Blöcken induziert worden sein. Nach (3.10) existiert aber auch x_j^p, da x_j in Klausel p auftritt. Nach (3.11) existiert somit die Kante $\{x_i^p, x_j^q\}$, d.h. R ist nicht unabhängig. □

Korollar 3.17. *Satz 3.16 gilt bereits, wenn die Größe der Blöcke auf* 3 *beschränkt ist.*

Beweis. Keiner der in (3.10) definierten Blöcke V_i enthält mehr als drei Knoten, da in jeder Klausel der Formel f maximal 3 Literale auftreten. □

Aus Satz 3.16 folgt nun das Hauptresultat dieses Abschnittes:

Satz 3.18. *Festzustellen, ob eine natürliche Zahl $s > 1$ eine untere Schranke für χ_P einer gegebenen PCP-Instanz ist, ist ein NP-vollständiges Problem.*

Beweis. Wenn $\chi_P(G) \geq s > 1$ gilt, kann es kein stabiles Repräsentantensystem geben. Nach Satz 3.16 ist es ein NP-vollständiges Problem, festzustellen ob ein solches System existiert, d.h. ob $\chi_P(G) = 1$ ist. Der Test für die untere Schranke löst dieses Problem implizit, ist also mindestens genauso schwer. □

3.4.3 Ein effizient lösbarer Spezialfall

Obwohl das Problem, ein stabiles Repräsentantensystem zu indentifizieren, NP-vollständig ist, kann es in bestimmten Graphen effizient gelöst werden. Dies ist zum Beispiel in Graphenklassen möglich, in denen unabhängig von der Blockstruktur alle stabilen Knotenmengen effizient bestimmt werden können und die Anzahl dieser Mengen nicht zu groß ist. Dann kann man alle stabilen Mengen aufzählen und jede daraufhin untersuchen, ob sie aus jedem Block einen Knoten enthält. Dies ist natürlich nur dann ein polynomielles Verfahren, wenn die Anzahl der unahbhängigen Mengen polynomiell in der Größe des Graphen beschränkt ist, eine exponentielle Anzahl solcher Mengen würde wieder zu einem ineffizienten Verfahren führen. Wir werden nun eine Graphenklasse vorstellen, bei der diese Voraussetzungen erfüllt sind.

Streng genommen muß nicht die Anzahl der stabilen Mengen polynomiell beschränkt sein, sondern es langt, wenn dies für die inklusionsmaximalen stabilen Mengen zutrifft. Dabei handelt es sich um die folgenden Mengen:

Definition 3.19. Eine Knotenmenge heißt *inklusionsmaximal* bezüglich einer Eigenschaft, wenn keine echte Obermenge diese Eigenschaft hat.

Es ist leicht einzusehen, daß diese schwächere Bedingung für unsere Zwecke ausreicht. Jede stabile Menge ist in einer inklusionsmaximalen stabilen Menge enthalten. Wenn also keine dieser maximalen stabilen Mengen alle Knoten überdeckt, tut dies auch keine der übrigen. Sollte eine stabile Menge jedoch alle Knoten überdecken, leistet dies auch jede Obermenge. Es langt also völlig, sich mit den maximalen stabilen Mengen zu beschäftigen.
Die inklusionsmaximalen unabhängigen Mengen in einem Graphen zu berechnen ist äquivalent zu dem Problem, die inklusionsmaximalen Cliquen in einem anderen Graphen zu finden, dem sogenannten Komplement des Graphen:

Definition 3.20 (Komplement eines Graphen)**.** Das Komplement eines Graphen $G = (V, E)$ ist der Graph

$$\overline{G} = (V, \{\{v, w\} \ : \ \{v, w\} \notin E\})$$

d.h. $\overline{G}$ enthält dieselben Knoten wie G und genau die Kanten, die nicht in G enthalten sind.

Lemma 3.21. *Die inklusionsmaximalen stabilen Mengen eines Graphen sind genau die inklusionsmaximalen Cliquen im Komplement des Graphen.*

Beweis. Zwischen den Knoten einer stabilen Menge existiert per Definition keine Kante. Folglich sind alle diese Kanten im Komplement enthalten, die Knoten bilden also dort eine Clique. Damit ist klar, daß jede stabile Menge im Komplement eine Clique bildet und umgekehrt, und folglich bleibt auch die Maximalität bezüglich Mengeninklusion erhalten. □

Definition 3.22. Ein Graph heißt *trianguliert* wenn es in jedem Kreis mit Länge größer als 3 eine Sehne gibt. Eine Sehne in einem Kreis $(v_1, \ldots, v_l)$ ist eine Kante $\{v_i, v_j\}$, deren Endpunkte im Kreis nicht benachbart sind.

Satz 3.23. *In einem triangulierten Graphen mit n Knoten gibt es höchstens n inklusionsmaximale Cliquen. Diese können effizient berechnet werden.*

Beweisskizze. Ein Graph ist genau dann trianguliert wenn er eine *Simplizialzerlegung* besitzt, d.h. eine Numerierung $v_1, \ldots, v_n$ der Knoten, so daß die Nachbarn von v_i, deren Nummer größer als i ist, eine Clique bilden. Eine solche Simplizialzerlegung kann in $O(|V| + |E|)$, also in Linearzeit, berechnet werden.
Die inklusionsmaximalen Cliquen eines solchen Graphen haben alle die Form

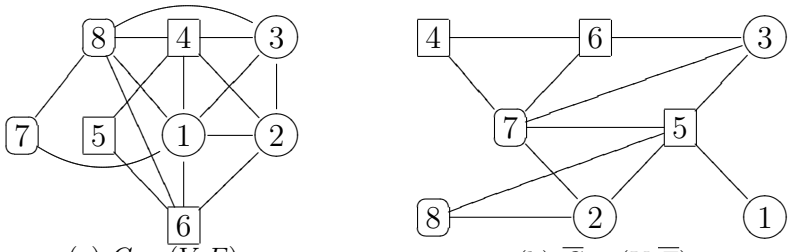
$$v_i \cup (\delta(v_i) \cap \{v_j \ : \ j \geq i\})$$

d.h. es gibt höchstens n maximale Cliquen, und diese können effizient bestimmt werden.
Eine genaue Ausführung mit vollständigen Beweisen findet man z.B. in [5], Kapitel 4. □

Zusammengefaßt ergeben die vorhergehenden Betrachtungen eine Graphenklasse, in der es, wie am Anfang des Abschnittes verlangt, nur eine begrenzte Anzahl inklusionsmaximaler stabiler Mengen gibt, die effizient aufgezählt werden können.

Satz 3.24. *Sei G ein partitionierter Graph, dessen Komplement trianguliert ist. Dann kann effizient festgestellt werden, ob in G ein stabiles Repräsentantensystem existiert.*

Beweis. Wenn $\overline{G}$ trianguliert ist, können dort nach Satz 3.23 die maximalen Cliquen effizient berechnet werden. Diese Cliquen im Komplement entsprechen nach Lemma 3.21 genau den maximalen stabilen Mengen im Ausgangsgraphen. Folglich gibt es in G höchstens n inklusionsmaximale stabile Mengen. Offensichtlich kann für jede dieser Mengen effizient getestet werden, ob sie mindestens einen Knoten aus jeder Partition enthält und somit ein stabiles Repräsentantensystem enthält. □



(a) $G = (V, E)$ (b) $\overline{G} = (V, \overline{E})$

Abbildung 3.5: Beispiel für einen Graphen mit trianguliertem Komplement

Beispiel. Abbildung 3.5 zeigt einen Graphen 3.5(a), dessen Komplement 3.5(b) trianguliert ist. Die Blöcke in diesem Graphen sind $\{1, 2, 3\}$, $\{4, 5, 6\}$ und $\{7, 8\}$. Im Komplement gibt es sechs maximale Cliquen, von denen die vier $\{2, 5, 7\}$, $\{2, 5, 8\}$, $\{3, 5, 8\}$ und $\{3, 6, 8\}$ alle Blöcke überdecken. Diese Cliquen entsprechen den stabilen Repräsentantensystemen im ursprünglichen Graphen.

Eine Klasse von Graphen, die diese Eigenschaft haben, sind die Komparabilitätsgraphen von Intervallordnungen. Die Knoten dieser Graphen entsprechen reellen Intervallen $[v_l^1, v_r^1], \ldots, [v_l^n, v_r^n]$. Die Kante $\{v_i, v_j\}$ existiert, wenn sich die entsprechenden Intervalle nicht überlappen. Im Komplement dieser Graphen sind zwei Knoten also folglich genau dann adjazent, wenn sich die Intervalle schneiden. Solche Graphen sind nach [5] trianguliert.

Kapitel 4

Heuristiken für das Partition Coloring Problem

Im vorangegangen Kapitel haben wir gezeigt, daß PCP NP-vollständig ist. Nach derzeitigem Stand der Forschung heißt dies, daß im allgemeinen Fall die Optimallösungen nicht effizient berechnet werden können. Doch selbst die Kenntnis eines optimalen Repräsentantensystems würde nur bedingt helfen, da schon bei verhältnismäßig kleinen Graphen eine optimale Färbung nicht mit zufriedenstellender Laufzeit bestimmt werden kann. Selbst die besten derzeit bekannten exakten Färbungsalgorithmen enumerieren im wesentlichen alle möglichen Färbungen und unterscheiden sich hauptsächlich darin, wie nicht erfolgversprechende Teilfärbungen erkannt und vermieden werden. Ein Beispiel für einen exakten Färbungsalgorithmus findet man in [1].
Für das Problem in partitionierten Graphen bedeutet dies, daß zusätzlich zur Färbung auch noch die möglichen Repräsentantensysteme aufgezählt werden müßten. Dieses Vorgehen ist nicht realistisch, da die Anzahl dieser Systeme im Vergleich zur Größe des Graphen enorm ist. Beispielsweise gibt es in einem relativ kleinen Graphen mit 25 Blöcken aus je zwei Knoten bereits $2^{25} \approx 3 \cdot 10^7$ Möglichkeiten, die Knoten zu wählen. Die Graphen, die beim Primerdesign entstehen, bestehen oft aus mehreren tausend Knoten, so daß sich die Enumeration praktisch verbietet.
Das Ziel ist also, Heuristiken zu entwickeln, die bei akzeptabler Laufzeit eine möglichst gute Lösung bestimmen. In diesem Kapitel werden wir die bisher existierenden Verfahren vorstellen und im Anschluß zwei neue Methoden vorstellen. Zusätzlich präsentieren wir ein effizientes Verfahren, mit dem Knoten identifiziert werden können, die in Heuristiken nicht berücksichtigt werden müssen.

4.1 bisher bekannte Heuristiken

In diesem Abschnitt möchten wir die bisher bekannten Heuristiken für das Partition Coloring Problem vorstellen. Die bisher bekannten Algorithmen wurden für ein Routing-Problem in optischen Netzwerken entwickelt. Das dafür entwickelte Modell führt auf dasselbe Modell wie das von uns untersuchte Primer-Design.
In optischen Netzwerken sind die Kommunikationspartner mit Glasfasern verbun-

den. Die Übertragung erfolgt über Licht einer vorher festgelegten Frequenz. Über eine Faser können mehrere Verbindungen gleichzeitig laufen, wenn sie unterschiedliche Frequenzen benutzen. In derartigen Netzwerken ist das Problem des Routings und der Frequenzzuweisung (RWA, "Routing and Wavelength Assignment"), für eine vorgegebene Menge von Verbindungen zwischen Netzwerkknoten die Wege durch das Netzwerk so zu wählen, daß möglichst wenige Frequenzen benutzt werden.
In der Praxis wird für jede Verbindungsanfrage eine gewisse Anzahl möglicher Wege durch das Netzwerk vorgegeben, da die Anzahl möglicher Routingwege sehr groß sein kann. Das Ziel ist, die Wege aus den gegebenen Mengen so auszuwählen, daß möglichst wenige Frequenzen benötigt werden. Dieser Ansatz führt zum Partition Coloring Problem. Hier entsprechen die Knoten des Graphen den möglichen Wegen, die Blöcke werden von allen gegebenen Wegen zu je einer Verbindung gebildet. Zwei Knoten sind genau dann verbunden, wenn die entsprechenden Wege zumindest eine Faser gemeinsam benutzen, also nicht dieselbe Frequenz benutzen können. Auch hier entspricht eine optimale Lösung des Färbungsproblems einer Optimallösung des RWA-Problems.

4.1.1 Greedyalgorithmen

Greedy für Graphenfärbungen

Der Greedy-Algorithmus ist die am weitesten verbreitete Methode um Graphenfärbungen zu berechnen. Das Prinzip des Greedy ist, die Knoten nacheinander in einer frei wählbaren Reihenfolge zu färben. Dabei wird dem Knoten stets die niedrigste zulässige Farbe zugewiesen. Wenn keine der bisher verwendeten Farben zulässig ist, wird eine weitere Farbe eingeführt und für diesen Knoten benutzt. Die Details dieses Verfahrens sind noch einmal in Algorithmus 1 ausgeführt. Dort werden die Knoten erst numeriert (Schritte 2 bis 5) und anschließend wie gerade beschrieben gefärbt (Schritte 6 bis 8)

Algorithmus 1 Greedyalgorithmus

Eingabe: Graph $G = (V, E)$
Ausgabe: Färbung von G
1: $i = 1$
2: **while** Es existiert $v \in V$ ohne Nummer **do**
3: wähle einen Knoten $v \in V$ ohne Nummer als v_i
4: $i = i + 1$
5: **end while**
6: **for** $i = 1, \ldots, n$ **do**
7: färbe v_i mit der kleinsten zulässigen Farbe
8: **end for**

Die Reihenfolge, in der die Knoten ausgewählt werden, hat bei diesem Vorgehen einen entscheidenden Einfluß auf das Ergebnis. Zum Beispiel kann der Graph in

Abbildung 4.1 mit zwei Farben gefärbt werden, wenn die Knoten in der Reihenfolge a, b, c, d gewählt werden. Dagegen werden bei der Reihenfolge a, d, b, c drei Farben verwendet.

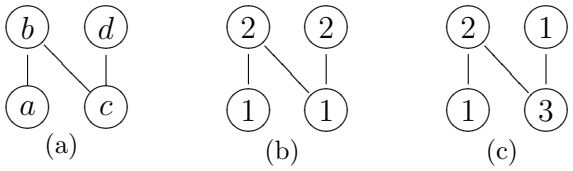


Abbildung 4.1: Unterschiedliche Auswahlstrategien ändern die Anzahl benötigter Farben

Die Knotenauswahl (im Algorithmus Schritt 3) ist also die kritische Komponente des Greedy-Verfahrens. Die folgenden Auswahlstrategien sind die am häufigsten verwendeten:

LargestFirst [15]: Ordne die Knoten absteigend nach Knotengrad. v_1 wird also ein Knoten mit maximalem Grad etc. Der Grad wird jeweils im gesamten Graphen ermittelt.

SmallestLast [11]: Wähle einen Knoten mit minimalem Knotengrad als v_i. Dabei wird nur der Grad in dem Teilgraphen betrachtet, der von den unnumerierten Knoten induziert wird. Abweichend von der Darstellung in Algorithmus 1 wird in umgekehrter Reihenfolge gefärbt, das heißt beginnend mit v_n bis v_1.

DSatur [1]: Bei dieser Variante wird ein Knoten sofort bei der Auswahl gefärbt, der Schritt 7 wird in den Schritt 3 integriert. Als v_i wird ein Knoten gewählt, der maximalen Farbgrad hat. Der Farbgrad eines Knotens ist die Anzahl unterschiedlicher Farben, mit denen seine Nachbarn gefärbt sind. Aus mehreren Kandidaten mit gleichem Farbgrad wird der Knoten mit den meisten ungefärbten Nachbarn ausgewählt.

Greedyalgorithmen für PCP

Li und Simha haben in [10] Adaptionen der oben beschriebenen Greedyverfahren für das Partition Coloring Problem entwickelt. Dabei ist die Grundannahme, daß die Wahl von Knoten mit kleinem Grad wahrscheinlich auch zu einer geringeren Farbzahl führt. Diese Annahme ist intuitiv einzusehen, da die chromatische Zahl eines Graphen den maximalen Knotengrad höchstens um eins überschreitet (siehe [9], Theorem 16.11). Der Grund dafür ist, daß mit dieser Farbzahl für jeden Knoten selbst dann eine Farbe zulässig ist, wenn alle seine Nachbarn unterschiedlich gefärbt sind.
Wenn ein Knoten aus einem Block gewählt worden ist, können die übrigen Knoten des Blockes entfernt werden, da diese nicht mehr betrachtet werden müssen. In

Kombination mit den oben beschriebenen Auswahlstrategien für das Greedyverfahren ergeben sich damit die folgenden Strategien:

LargestFirst: In jedem Schritt soll der Knoten mit dem "größten kleinsten Grad" ausgewählt werden. Man bestimmt dazu zunächst aus jedem Block, für den noch ein Repräsentant gesucht wird, einen Kandidaten und bestimmt anschließend aus diesen den nächsten Repräsentanten. Als Kandidaten wählt man aus jedem Block einen Knoten mit möglichst kleinem Knotengrad. Aus diesen vorgewählten Knoten wählt man den mit dem größten Grad aus. Sollten bei einem dieser Auswahlschritte mehrere Wahlen möglich sein, kann ein beliebiger Knoten ausgesucht werden. Dem ausgewählten Knoten weist man die kleinste zulässige Farbe zu und entfernt alle anderen Knoten aus dem entsprechenden Block.Diese beiden Auswahlschritte werden wiederholt, bis für alle Blöcke ein Repräsentant ausgewählt und gefärbt worden ist.

SmallestLast: Hier entfällt der doppelte Auswahlschritt, da beide Male der kleinere Grad entscheiden soll. Es muß also aus den Knoten aller Blöcke einer mit minimalem Grad gewählt werden. Anschließend kann der gesamte entsprechende Block, auch der gewählte Knoten, aus dem Graphen entfernt werden. Wenn alle Repräsentanten so gewählt worden sind, erhalten die ausgesuchten Knoten in umgekehrter Reihenfolge der Auswahl die jeweils kleinste mögliche Farbe.

DSatur: Das Vorgehen ist ähnlich wie bei LargestFirst. Aus jedem Block, für den noch kein Repräsentant gewählt wurde, wird ein Knoten mit minimalem Farbgrad gesucht. Wenn es mehrere Kandidaten innerhalb eines Blockes gibt, entscheidet die kleinere Anzahl ungefärbter Nachbarn. Aus diesen Knoten wählt man den mit dem größten Farbgrad, bei Gleichheit entscheidet in diesem Schritt die größere Anzahl ungefärbter Nachbarn. Dem so gewählten Knoten wird die kleinste zulässige Farbe zugewiesen und alle anderen Knoten des Blockes entfernt.

Zusätzlich zu diesen Algorithmen haben Li und Simha auch die Varianten des Greedyalgorithmus untersucht, bei denen man erst die Repräsentanten wählt und anschließend färbt. Ausgehend von der oben beschriebenen Annahme, daß Knoten kleinen Grades die besten Repräsentanten sind, gehen diese Verfahren so vor:
Zunächst wird ein Knoten kleinsten Grades gewählt. Alle anderen Knoten aus dem Block, in dem dieser Knoten liegt, werden aus dem Graphen entfernt. Dann wird aus den übrigen Blöcken wieder ein Knoten mit minimalem Grad gewählt und so fort, bis aus jedem Block ein Knoten gewählt worden ist. Der Graph besteht dann nur noch aus den gewählten Repräsentanten. Im zweiten Schritt wird die Färbung des von den gewählten Knoten induzierten Teilgraphen bestimmt, indem eine der im vorangegangen Abschnitt beschrieben Greedy-Strategien benutzt wird.

4.1.2 Tabu-Search

Eine weitere Heuristik für das Partition Coloring Problem haben Noronha und Ribeiro in [3] vorgestellt. Sie benutzen die Tabu-Search Technik, um ausgehend von einer gegebenen Lösung des Problems eine Lösung zu finden, die eine Farbe weniger benötigt.
Das Prinzip von Tabu-Search ist kurz gesagt folgendes: Ausgehend von einer zulässigen Lösung eines Problems wird eine gewisse Menge ähnlicher ("benachbarter") Lösungen untersucht, und die beste davon akzeptiert, selbst wenn sie eine Verschlechterung darstellt. Anschließend wird für eine gewisse Anzahl Iterationen verboten, die dabei an der Lösung vorgenommenen Modifikationen rückgängig zu machen. Diese Verbote werden in der sogenannten Tabu-Liste gespeichert, von der das Verfahren seinen Namen hat. Mit dieser Liste soll verhindert werden, daß man zu einer bereits untersuchten Lösung zurückkehrt.
Der Erfolg dieser Methode hängt entscheidend von drei Faktoren ab: den generierten Nachbarschaften einer Lösung, der Anzahl Iterationen und der Verweildauer von Verboten in der Tabu-Liste. Diese Parameter müssen jeweils auf die konkrete Anwendung abgestimmt werden.

Das Ziel des Verfahrens von Noronha und Ribeiro ist, wie eingangs erwähnt, eine Farbe gegenüber einer gegebenen Lösung einzusparen. Dazu wird allen Knoten einer Farbe zufällig eine der übrigen Farben zugewiesen. Die so entstandene Färbung wird im allgemeinen nicht zulässig sein, sondern Konflikte, also adjazente Knoten mit der selben Farbe, enthalten. An dieser Stelle setzt die Tabu-Suche an, um die Konflikte aufzulösen.
Um die benachbarten Färbungen zu erzeugen sucht man in allen Blöcken, die in der aktuellen Färbung von Konflikten betroffen sind, nach Verbesserungen. Sei B die Menge dieser Blöcke. Man beginnt in einem dieser Blöcke und probiert statt des Repräsentanten und dessen Farbe in der aktuellen Lösung alle nicht verbotenen Kombinationen aus Knoten und Farbe in diesem Block aus. Wenn sich dabei eine Färbung mit weniger Konflikten ergibt, wird diese die neue aktuell beste Färbung. B wird aktualisiert und die Suche beginnt von neuem in den entsprechenden Blöcken. Sonst fährt man mit dem nächsten Block aus B fort. Wenn alle Blöcke aus B untersucht worden sind, ohne eine Färbung mit weniger Konflikten zu finden, wählt man aus allen in dieser Iteration untersuchten neuen Färbungen die mit den wenigsten Konflikten. Immer wenn eine neue Färbung gewählt wird, werden alle Paare aus Knoten und Farbe aus der Färbung davor in die Tabu-Liste aufgenommen, um nicht zu der vorigen Lösung zurückzukehren.
Wenn sich im Laufe der Suche eine Lösung ohne Konflikte ergibt, wird die Tabu-Liste geleert und das Verfahren beginnt von vorne, um eventuell eine weitere Farbe einzusparen. Das gesamte Verfahren wird beendet, wenn insgesamt $q(c_{\max}-1)\cdot F_{end}$ Färbungen untersucht worden sind, wobei q die Anzahl der Blöcke, $c_{\max}$ die Anzahl benutzter Farben in der Startlösung und F_{end} ein wählbarer Faktor ist.

4.2 Preprocessing

Ein erster Schritt bei der Entwicklung von Heuristiken ist zu prüfen, ob Teile des Graphen nicht in die Suche einbezogen werden müssen. Damit wird erreicht, daß zum einen die notwendige Datenmenge und somit auch der Ressourcenbedarf möglichst klein bleibt, zum anderen müssen keine Knoten betrachtet werden, die nicht in einer Optimallösung auftreten werden.
Es ist offensichtlich, daß Kanten innerhalb eines Blockes bei der Suche nach einem Repräsentantensystem nicht beachtet werden müssen. Weil aus jedem Block nur je ein Knoten gewählt wird, kann keine dieser Kanten in dem von der Auswahl induzierten Teilgraphen vorkommen. Diese Kanten wegzulassen kann allerdings die Suchzeiten in den Nachbarmengen von Knoten drastisch reduzieren. Im weiteren gehen wir deshalb davon aus, daß die Knoten eines Blockes paarweise unabhängig sind, d.h. nicht durch eine Kante verbunden sind.
Zusätzlich wollen wir Knoten identifizieren, die nicht betrachtet werden müssen. In diesem Abschnitt werden wir ein Kriterium vorstellen, wann ein Knoten aus dem Graphen entfernt werden kann, sowie einen effizienten Algorithmus um diese Knoten zu finden.

4.2.1 Knoten mit nichtminimaler Nachbarmenge

Ein Knoten muß in den Berechnungen nicht betrachtet werden, wenn in jeder optimalen Lösung, die diesen Knoten enthält, ein anderer Repräsentant für den entsprechenden Block gewählt werden kann, ohne daß die Anzahl benötigter Farben steigt. Selbstverständlich muß entweder dieser oder einer der alternativen Knoten im Graphen erhalten bleiben, um diese Lösung nicht zu verbieten.
Ein hinreichendes Kriterium für eine solche Konstellation ist, wenn es zu einem Knoten w einen weiteren Knoten v im selben Block gibt, dessen Nachbarmenge $\delta(v)$ eine Teilmenge der Nachbarn von w ist. In diesem Fall kann in jedem Repräsentantensystem, das w enthält, v als neuer Repräsentant gewählt werden. Die Farbe von w ist auch für v zulässig, da v nur zu Knoten adjazent ist, die auch Nachbarn von w sind. Für diese Situation schreiben wir $v \prec w$.

Definition 4.1. Sei $G = (V := V_1 \dot\cup \ldots \dot\cup V_k, E)$ und $V_i = \{v_{i1}, \ldots, v_{il}\}$. Seien v_{ip} und v_{iq} zwei unterschiedliche Knoten aus V_i. Wir schreiben

$$v_{ip} \prec v_{iq} \iff \begin{array}{rl} \text{entweder} & \delta(v_{ip}) \subsetneq \delta(v_{iq}) \\ \text{oder} & \delta(v_{ip}) = \delta(v_{iq}) \text{ und } p < q \end{array} \tag{4.1}$$

Insbesondere gilt $u \prec v, v \prec w \Rightarrow u \prec w$. Die Ausnahme für den Fall, daß die Nachbarmengen gleich sind, dient dazu, den Fall $v \prec w$ und gleichzeitig $w \prec v$ zu verbieten. So wird die Relation "$\prec$" auf eine beliebige, aber feste Weise auf Knoten mit gleicher Nachbarmenge fortgesetzt.
Wenn $v \prec w$ gilt, heißt v *unterer Nachbar* von w bzgl. "$\prec$", entsprechend w *oberer*

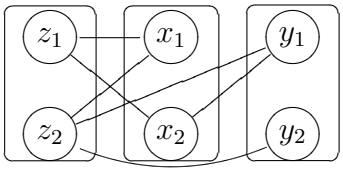


Abbildung 4.2: Beispiel für nichtminimale Knoten in $G_\prec$. Es gilt $y_2 \prec y_1$, erst nach entfernen von y_1 auch $x_2 \prec x_1$

Nachbar von v. Ein Knoten heißt *minimal* beziehungsweise *maximal* bzgl. "$\prec$", wenn er keinen unteren bzw. oberen Nachbarn hat.

In dieser Notation sind die Knoten, die nach der Überlegung oben aus dem Graphen entfernt werden können, genau die, die bezüglich "$\prec$" nicht minimal sind. Wir definieren daher den Graphen, in dem nur noch die minimalen Knoten enthalten sind:

Definition 4.2 (reduzierter Graph)**.** Sei G wie in Definition 4.1. Wir definieren

$$V_\prec := \{v \in V \ : \text{ es existiert kein } v' \text{mit } v' \prec v\}$$
$$G_\prec := (V_\prec, E(V_\prec))$$

$G_\prec$ enthält also alle bezüglich "$\prec$" minimalen Knoten und heißt der *reduzierte Graph* zu G. Die Sonderbehandlung von $\delta(v) = \delta(w)$ in Definition 4.1 führt hier dazu, daß aus einer Gruppe von Knoten mit identischer Nachbarmenge genau einer gewählt wird.

In dem so definierten Graphen kann es allerdings wieder bzgl. $\prec$ nicht minimale Knoten geben. Diese Situation illustriert Abbildung 4.2. Daher kann man zum reduzierten Graphen wieder dessen reduzierten Graphen berechnen etc. Damit erhält man eine Folge von immer kleiner werdenden Graphen, die ineinander enthalten sind. Es kann allerdings nur eine begrenzte Anzahl derartiger Graphen geben:

Satz 4.3. *Sei $G^0_\prec = G$ und $G^j_\prec = (G^{j-1}_\prec)_\prec$. Dann gibt es ein kleinstes $l \in \mathbb{N}$, so daß $G^l_\prec = G^i_\prec$ für alle $i \geq l$.*

Beweis. Der Satz folgt unmittelbar aus der Tatsache daß $|V(G^i_\prec)| \leq |V(G^{i+1}_\prec)|$, wobei Ungleichheit genau dann gilt wenn $G^i_\prec \neq G^{i+1}_\prec$. Durch die Vorschrift für $\delta(v) = \delta(w)$ wird sichergestellt, daß mindestens ein Knoten in jedem Block übrig bleibt. Somit gibt es nur endlich viele so konstruierte Graphen $G^i_\prec$. □

Definition 4.4. Der Graph $G^l_\prec$ aus Satz 4.3 heißt der *vollständig reduzierte Graph* zu G.

4.2.2 Ein Algorithmus zum Entfernen nichtminimaler Knoten

Es ist naheliegend, sukzessive die reduzierten Graphen $G^i_{\prec}$ aus Satz 4.3 zu berechnen. Dazu müßten bei naivem Vorgehen in jeder Iteration alle Nachbarmengen von Knoten innerhalb eines Blockes miteinander verglichen werden. Dieses unter Umständen recht aufwendige Verfahren ist jedoch vermeidbar. Betrachten wir noch einmal den Graphen in Abbildung 4.2. x_1 kann aus $G_{\prec}$ entfernt werden, weil mit y_1 der einzige nicht auch zu x_2 adjazente Nachbar von x_1 wegfällt. Dies ist auch die einzige Konstellation, in der weitere Knoten entfernt werden können, wie die folgende Charakterisierung zeigt.

Lemma 4.5. *Sei $v \prec w$. In $G' = (V' = V \setminus w, E(V'))$ gilt $x \prec y$ für zwei Knoten $x, y \neq w$ genau dann, wenn*

$$\begin{aligned} & x \prec y \text{ in } G \\ \text{oder} \quad & x \prec y \text{ in } G' \text{ und } \{w\} = \delta(x) \setminus \delta(y) \text{ in } G \end{aligned} \tag{4.2}$$

Das heißt, daß durch das Entfernen von w nur solche Knoten y nichtminimal werden können, für die ein x existiert bei dem $x \prec y$ nur wegen $w \in \delta(x)$ nicht gegolten hat.

Beweis. Wenn schon in G $x \prec y$ gilt, muß dies in G' immer noch der Fall sein. Entweder war w weder zu x noch zu y adjazent, dann gilt weiter $x \prec y$. Sonst war w wegen $\delta(x) \subset \delta(y)$ auf jeden Fall zu y adjazent. Selbst wenn w kein Nachbar von x war gilt weiter $x \prec y$, da y immer noch mit allen Nachbarn von x verbunden ist. Wenn $x \prec y$ erst in G' gilt, kann in G noch nicht $\delta(x) \subset \delta(y)$ gewesen sein. Da in G' nur w fehlt, kann außer w kein weiterer Knoten zu x, aber nicht zu y, adjazent sein, das heißt gerade $\{w\} = \delta(x) \setminus \delta(y)$. □

Im Folgenden werden wir einen Algorithmus vorstellen, der den vollständig reduzierten Graphen berechnet. Dabei wird mit Hilfe von Lemma 4.5 die Menge der zu überprüfenden Knoten eingeschränkt, um möglichst wenige Mengenvergleiche durchführen zu müssen.
Sei M die Menge aller Knoten v, für die $v \prec w$ für ein geeignetes w aus demselben Block gelten kann. Zu Beginn ist $M = V$, da keine Informationen vorhanden sind. Um entsprechende w zu finden wird für jeden Knoten des Blockes getestet, ob seine Nachbarn eine Obermenge der Nachbarn von v sind. Wird ein solches w gefunden, wird es aus dem Graphen entfernt. Nach Lemma 4.5 sind die Nachbarn von w die einzigen Knoten, die nun zusätzliche obere Nachbarn haben können, und werden deshalb wieder in M eingefügt. Solange M nicht leer ist wird ein v gewählt und nach entsprechenden Knoten w gesucht. In Algorithmus 2 wird das Vorgehen noch einmal ausführlich dargestellt.

Satz 4.6. *Sei $|V| = n$. Algorithmus 2 berechnet den vollständig reduzierten Graphen zu G. Die Laufzeit ist $O(n^5)$.*

Algorithmus 2 Preprocessing

Eingabe: $G = (V := V_1 \dot\cup \ldots \dot\cup V_k, E)$

```
   Sei block(v) = V_i : v ∈ V_i
 1: Initialisiere eine Menge M = V
 2: while M ≠ ∅ do
 3:     wähle v ∈ M
 4:     M = M \ v
 5:     for all w ∈ block(v) do
 6:         if δ(w) ⊃ δ(v) then
 7:             block(v) = block(v) \ w
 8:             M = M ∪ δ(w)
 9:         end if
10:     end for
11: end while
```

Lemma 4.7. *Die Reihenfolge, in der die nichtminimalen Knoten entfernt werden, beeinflußt das Ergebnis bis auf Isomorphie nicht.*

Beweis. Wenn für zwei Knoten $v \prec w$ gilt, also w einen unteren Nachbarn hat, wird dies nach Lemma 4.5 durch das Entfernen von nichtminimalen Knoten nicht verändert. Sollte durch die Löschoperation zusätzlich $\delta(w) \subset \delta(v)$ gelten, dann kann einer dieser beiden Knoten entfernt werden. Sollte v entfernt werden, da $u \prec v$ gilt, dann war $u \prec v \prec w$ und somit auch $u \prec w$. Ein nichtminimaler Knoten kann diese Eigenschaft also nicht mehr verlieren.
Es bleibt noch zu zeigen, daß ein Knoten, der durch eine bestimmte Reihenfolge von Löschoperationen nichtminimal wird, dies auch bei jeder anderen Reihenfolge wird. Sei also z ein Knoten, der bei einer Reihenfolge nichtminimal wird, bei einer anderen jedoch nicht. Wir betrachten die erste Elimination der ersten Reihenfolge, die in der zweiten nicht durchgeführt wird. Nach Lemma 4.5 wäre diese jedoch möglich, wenn alle vorangegangenen Eliminationen der ersten Reihenfolge durchgeführt worden sind. Daraus folgt, daß z auch in der zweiten Reihenfolge als nicht minimal erkannt wird, im Widerspruch zur Annahme. □

Beweis von Satz 4.6. Zum Beweis der Korrektheit des Verfahrens zeigen wir, daß zu jedem Zeitpunkt die Menge M alle Knoten enthält, die einen oberen Nachbarn bzgl. $\prec$ haben können. Bei der Initialisierung gilt dies auf jeden Fall. Wenn in Schritt 7 ein Knoten w entfernt wird, werden alle seine Nachbarn in M aufgenommen. Nach Lemma 4.5 sind dies alle Knoten, die im dadurch entstandenen Graphen zusätzliche obere Nachbarn haben können. Somit gilt die Behauptung für M weiter. Da nach Lemma 4.7 die Reihenfolge der Operationen das Ergebnis nicht beeinflußt, folgt die Korrektheit des Verfahrens.
Der Algorithmus muß terminieren, da M nur dann wächst, wenn ein Knoten aus

dem Graphen entfernt wurde. Da mindestens ein Knoten pro Block übrig bleibt, kann dies nur endlich oft passieren, d.h. das Verfahren endet nach endlicher Zeit.

Zur Laufzeit. Der Graph wird in sogenannten Adjazenzlisten gespeichert, d.h. für jeden Knoten existiert eine Liste seiner Nachbarn. Weiterhin seien die Knoten auf eine beliebige aber feste Weise geordnet und die Listen entsprechend sortiert. Dies ist mit einem einmaligen Aufwand von $O(n \log n)$ pro Knoten, insgesamt also $n^2 \log n$ möglich. Mit sortierten Listen kann in Linearzeit geprüft werden, ob eine Liste Teilmenge der anderen ist.
Ein Knoten kann mit dieser Datenstruktur in $O(n^2)$ aus dem Graphen entfernt werden. Es müssen schlimmstenfalls alle n Listen mit Länge $O(n)$ durchsucht werden, das eigentliche Löschen geht in konstanter Zeit. Damit ergibt sich insgesamt $O(n^2)$.
Die Menge M kann als (2,4)-Baum implementiert werden, so daß alle Operationen wie Suchen, Einfügen und Löschen in $O(\log n)$ durchgeführt werden können.
Insgesamt sind somit für jedes w in der Schleife 5 bis 10 $O(n)$ Operationen für den Listenvergleich, eventuell $O(n^2)$ für einen Löschvorgang und $n \log n$ für die Aktualisierung von M notwendig, wobei die Größe der Adjazenzmenge von w mit n abgeschätzt wird. Für eine Iteration der Schleife 2 bis 11 sind folglich

$$\log n + n(O(n) + O(n^2) + O(n \log n)) = O(n^3)$$

Operationen nötig.
Wie oft wird die Schleife 2 bis 11 durchlaufen? Ein Knoten kann höchstens $n-k+1$-mal in die Menge M aufgenommen werden. Einmal wird er bei der Initialisierung eingefügt, und im schlimmsten Fall auch jedes Mal, wenn in Schritt 7 ein Knoten entfernt wird. Da es n Knoten gibt und mindestens k übrig bleiben, kann es höchstens $n - k$ Löschvorgänge geben. Jeder Knoten wird also maximal $(n - k + 1)$-mal in M eingefügt. Da es n Knoten gibt, sind folglich höchstens $n(n - k + 1)$ Iterationen der Schleife möglich. Daraus ergibt sich eine Gesamtzahl von

$$n\ (n - k + 1)\ O(n^3) = O(n^5)$$

Operationen für den gesamten Algorithmus. □

4.3 Zwei einfache Heuristiken für PCP

Unsere neue Heuristik für das Partition Coloring Problem berechnet nicht selbständig eine Lösung, sondern versucht, eine bereits vorhandene Lösung zu verbessern. Das Ziel ist selbstverständlich, eine neue Lösung zu finden, für die weniger Farben benötigt werden. Damit die Laufzeit akzeptabel bleibt, werden nur einzelne lokale Veränderungen gesucht. Im Folgenden werden wir zunächst das Kernstück der Heuristiken beschreiben, einen Algorithmus, der in jeweils einem Block die Auswahl beziehungsweise Färbung anpaßt. Im Anschluß daran zeigen wir, wie auf dieser Grundlage Verfahren zur Verbesserung einer Lösung von PCP aussehen können.

4.3.1 Prinzip und Notationen

In diesem Abschnitt werden wir die folgenden Bezeichnungen benutzen

- der Graph hat die Struktur $G = (V = V_1 \dot\cup \cdots \dot\cup V_k, E)$, insbesondere wird mit k die Anzahl der Blöcke bezeichnet.
- zu G gehört das Repräsentantensystem $\mathcal{C}$, also $|\mathcal{C} \cap V_i| = 1$ für alle $1 \leq i \leq k$.
- Der Knoten v hat die Farbe $c(v)$. Die Knoten, die das Repräsentantensystem $\mathcal{C}$ bilden, sind mit einer Farbe aus dem Bereich $1, \ldots, c_{\max}$ gefärbt, die übrigen mit Farbe 0. Ein Knoten mit Farbe 0 wird auch als ungefärbt bezeichnet.
- Für $V' \subset V$ setzen wir $c(V') = \{c(v) \ : \ v \in V' \cap \mathcal{C}\}$, d.h. $c(V')$ ist die Menge der Farben, mit denen benachbarte Repäsentanten gefärbt sind.
- Es gilt $\mathcal{C} = C_1 \cup \cdots \cup C_{c_{\max}}$ so daß $C_i = \{v \in \mathcal{C} \ : \ c(v) = i\}$
- Wie beim Preprocessing (Algorithmus 2) ist $\text{block}(v) = V_i : v \in V_i$.

Das Ziel ist es, durch einfache und vor allem schnell ausführbare Änderungen am Repräsentantensystem oder dessen Färbung zu einer besseren Färbung zu gelangen. Bei diesen Anpassungen betrachten wir immer nur einen einzigen Block und beschränken uns dort auf die folgenden beiden Operationen

1. den gewählten Knoten umfärben, ohne das Repräsentantensystem zu ändern
2. für den Block einen anderen Repräsentanten zu wählen.

Das genaue Vorgehen ist, nach Möglichkeit dem Repräsentanten des gerade betrachteten Blockes eine andere Farbe zuzuweisen. Dies ist genau dann möglich, wenn für die gefärbten Nachbarn maximal $c_{\max} - 2$ Farben benutzt werden. Eine Farbe wird bereits für den Knoten benutzt, und eine andere als diese muß zulässig sein. Wenn für den gerade betrachteten Knoten keine weitere Farbe zulässig ist, suchen wir in diesem Block nach einem Knoten, dem eine andere Farbe zugewiesen werden kann. Wenn ein solcher Knoten gefunden wird, wird er mit der kleinsten zulässigen Farbe gefärbt und als neuer Repräsentant des Blockes gewählt. Die Suche in diesem Block wird nicht fortgesetzt. Wichtig ist insbesondere, daß keine neuen Farben eingeführt werden. Algorithmus 3 zeigt noch einmal die Details dieses Verfahrens. Der Algorithmus benötigt drei Parameter: den aktuellen Repräsentanten v des Blockes, die Anzahl der derzeit benutzten Farben $c_{\max}$, und natürlich das aktuelle Repräsentantensystem $\mathcal{C}$ wie oben beschrieben. Die Rückgabe ist `true` wenn eine Änderung vorgenommen worden ist, sonst `false`.

Satz 4.8. *Die Laufzeit des* RecolorOrMove*-Algorithmus ist*

$$O\left(\sum_{w \in \text{block}(v)} (|\delta(w)| + c_{max})\right)$$

Algorithmus 3 RecolorOrMove

Eingabe: Knoten v, höchste verwendete Farbe $c_{\max}$, Repräsentanten $\mathcal{C}$
Ausgabe: `true` wenn für $block(v)$ eine andere Farbe benutzt werden kann, sonst `false`

```
1:  function RecolorOrMove(v, c_max, C)
2:      if |c(δ(v))| ≤ c_max − 2 then        # ist für v eine weitere Farbe zulässig?
3:          C_c(v) = C_c(v) \ v
4:          c(v) = min ({1, ..., c_max} \ c(δ(v) ∪ v))
5:          C_c(v) = C_c(v) ∪ v
6:          return true
7:      else                                 # existiert ein v' für das eine Farbe außer c(v) zulässig ist?
8:          if ∃v' ∈ block(v) mit |c(δ(v')) ∪ c(v)| ≤ c_max − 1 then
9:              C_c(v) = C_c(v) \ v, c(v) = 0
10:             c(v') = min({1, ..., c_max} \ c(δ(v') ∪ v))
11:             C_c(v') = C_c(v') ∪ v'
12:             return true
13:         end if
14:     end if
15:     return false
16: end function
```

Beweis. Für jeden Knoten aus block(v) müssen die Farben seiner Nachbarn festgestellt werden. Das Durchmustern alles Adjazenzlisten eines Knotens w dauert $|\delta(w)|$. Die Suche nach einer möglichen Farbe kann anschließend in $O(|c_{\max}|)$ durchgeführt werden, wenn bei der Suche in der Adjazenzliste bereits die benutzten Farben abgespeichert worden sind. □

4.3.2 Lokale Verfeinerung

Der RecolorOrMove-Algorithmus bestimmt den Repräsentanten eines einzelnen Blockes. Ähnlich wie bei den Greedy-Algorithmen (Abschnitt 4.1.1) entscheidet auch hier die Reihenfolge, in der diese Anpassungen vorgenommen werden, über die Qualität des Ergebnisses. Das Ziel ist folglich, eine Abfolge der Repräsentanten zu wählen, so daß durch die wiederholte Anwendung von RecolorOrMove die Farbzahl möglichst stark und schnell reduziert wird.
Die Idee des hier vorgestellten Verfahrens ist, nacheinander alle Knoten einer Farbe zu wählen, um so möglichst diese Farbe einzusparen. Sollte dies nicht erfolgreich sein, wählt man die nächste Farbe aus, bis eine Verbesserung erzielt worden ist. Es ist allerdings wahrscheinlich, daß bei diesem Vorgehen nicht sofort eine Farbe eingespart werden kann. In den meisten Fällen wird es notwendig sein, in Blöcken mit unterschiedlich gefärbten Repräsentanten Veränderungen durchzuführen, um zu ei-

ner besseren Lösung zu gelangen. Von daher sollte jeder Block mehrmals untersucht werden. Am leichtesten erreicht man dies, indem man wieder von vorne beginnt nachdem alle Farbklassen einmal untersucht worden sind.
Um die Aussicht, Fortschritt zu machen, weiter zu steigern, wählen wir die Farbklassen nicht beliebig aus, sondern nach steigender Größe. Dahinter steht die Vermutung, daß es einfacher ist, in wenigen Blöcken entsprechende Repräsentanten zu finden als in vielen. Innerhalb einer Farbklasse betrachten wird die Repräsentanten nicht in einer bestimmten Reihenfolge. Es empfiehlt sich, hier eine Abfolge zu wählen, die möglichst schnell zu implementieren ist.
Zu dem Verfahren wird natürlich noch ein Abbruchkriterium benötigt. Wir haben uns dafür entschieden die Suche abzubrechen, wenn eine bestimmte Anzahl von Suchvorgängen in allen Farbklassen erfolglos waren. Dieser Parameter entscheidet über die Qualität und die Laufzeit des Verfahrens und ist daher die kritische Größe. Auf die Wahl dieses Parameters werden wir bei der Analyse der so erzielten Ergebnisse in Kapitel 5 genauer eingehen.
Das vollständige Verfahren wird in Algorithmus 4 noch einmal technisch ausgeführt.

Algorithmus 4 Lokale Verfeinerung

```
 1: procedure RefineLocal(G, C = C_1 ∪̇ ··· ∪̇ C_{c_max})
 2:     count = 0
 3:     while count < MaxIter do
 4:         for i = c_max, ..., 1 do
 5:             for all v ∈ C_i do RecolorOrMove(v, c_max, C)
 6:             end for
 7:             if C_i = ∅ then
 8:                 vertausche Farben i und c_max
 9:                 lösche C_{c_max}
10:                 c_max = c_max − 1
11:                 Sortiere C so daß wieder |C_1| ≥ ··· ≥ |C_{c_max}|
12:                 count = 0
13:                 goto 3
14:             end if
15:         end for
16:         count = count + 1
17:         Sortiere C so daß wieder |C_1| ≥ ··· ≥ |C_{c_max}|
18:     end while
19: end procedure
```

Ein Durchlauf durch alle Farbklassen findet in der for-Schleife von Schritt 4 bis 15 statt. Wir gehen davon aus, daß zu Beginn jeder Iteration $|C_1| \geq \cdots \geq |C_{c_{\max}}|$ gilt, so daß $C_{c_{\max}}$ die Farbklasse mit der kleinsten Kardinalität ist. Innerhalb dieser Schleife wird für alle Repräsentanten, die mit der aktuellen Farbe gefärbt sind, RecolorOrMove aufgerufen, um nach im Sinne der Heuristik besseren Repräsentanten zu

suchen. Die Repräsentanten mit Farbe i sind nach Definition gerade die Knoten in C_i. Die Anzahl nicht erfolgreicher Iterationen über alle Farbklassen wird im Zähler "count" festgehalten.
Wenn die Farbe i nicht mehr benutzt wird, also $C_i = \emptyset$ gilt, werden die Farben i und $c_{\max}$ vertauscht und die jetzt leere Farbklasse $C_{c_{\max}}$ gelöscht. Damit wird sichergestellt, daß die benutzten Farben aufeinanderfolgende Nummern haben. $c_{\max}$ muß anschließend selbstverständlich vermindert werden. Dann wird die Sortierung der Farbklassen wiederhergestellt, damit die nächste Iteration wieder mit den richtigen Startbedingungen beginnt. Schließlich wird der Iterationszähler "count" zurückgesetzt und ein neuer Durchlauf durch die Farbklassen gestartet, bei dem wieder mit der Farbklasse kleinster Kardinalität $C_{c_{\max}}$ begonnen wird. Das Rücksetzen des Zählers soll sicherstellen, daß die Chancen auf eine weitere Verbesserung nicht durch eine zu geringe verbleibende Anzahl Iterationen vermindert werden.
Wenn keine Farbe eingespart werden kann, wird nur der Zähler erhöht und die Sortierung der Farbklassen wieder angepaßt.

Bei dem hier vorgestellten Verfahren werden die Veränderungen aus Iterationen, in denen die Farbzahl nicht gesunken ist, nicht rückgängig gemacht. Damit soll erreicht werden, daß nicht immer wiederkehrende Konfigurationen untersucht werden und sich blockierende Repräsentanten über mehrere Iterationen hinweg aufgelöst werden können (siehe dazu auch Abbildung 4.3). Die Minimumbildung bei der Farbauswahl aus RecolorOrMove sorgt außerdem dafür, daß die Farben zu den großen Farbklassen hin verändert werden. Das Ziel hiervon ist, auch in nicht erfolgreichen Iterationen eine Art Fortschritt zu machen, also die Kardinalität der kleineren Farbklassen weiter zu verringern.

4.3.3 Rekursive Verfeinerung

Das im vorangegangenen Abschnitt beschriebene Verfahren ist nicht die einzige Möglichkeit, wie auf Basis des RecolorOrMove-Algorithmus eine Lösung für PCP berechnet werden kann. Bereits bei der Beschreibung der lokalen Verfeinerung haben wir erwähnt, daß eine Verringerung der Farbzahl unter Umständen nur durch Veränderungen in verschiedenen Farbklassen erreicht werden kann. Ein Beispiel für eine solche Konstellation ist der Graph in Abbildung 4.3. Die Repräsentanten bilden ein Dreieck, benötigen also drei Farben. Beim Repräsentanten des mittleren Blockes würde RecolorOrMove nicht funktionieren. Dafür kann im rechten Block der andere Knoten als Repräsentant gewählt werden und die Farbe 2 erhalten. Dadurch wird für den Knoten im mittleren Block die Farbe 2 zulässig. In diesem Abschnitt beschreiben wir eine Auswahlstrategie, die auf dieser Beobachtung basiert.
Die grundsätzliche Technik, zuerst in den Farbklassen kleiner Kardinalität zu suchen, bleibt. Wenn jedoch in einem Block RecolorOrMove scheitert, kommt ein rekursives Auswahlverfahren zum Einsatz, um den Knoten eventuell doch noch umfärben zu können.

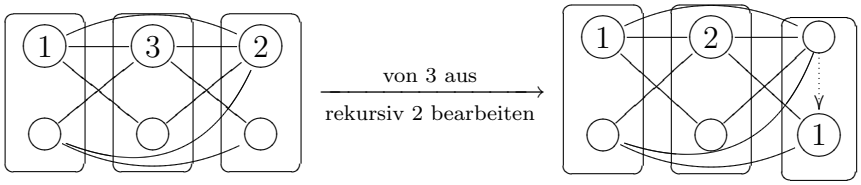


Abbildung 4.3: Beispiel für die Verminderung der Farbzahl durch Operationen in zwei Blöcken

RecolorOrMove schlägt fehl, wenn für keinen Knoten des betrachteten Blockes eine andere Farbe als die des aktuell gewählten Repräsentanten zulässig ist. Folglich sind alle diese Knoten zu Knoten aller anderen Farben adjazent. Daraus folgt, daß eine Veränderung in diesem Block erst möglich sein wird, wenn diese Situation für mindestens einen Knoten aufgelöst werden kann.
Dazu betrachten wir nun eine beliebige Farbe außer der des aktuellen Knotens, und alle Blöcke mit Repräsentanten in der gewählten Farbe, die zu diesem Knoten adjazent sind. Wenn man durch lokale Anpassung in diesen Blöcken jeweils eine andere Farbe benutzen kann, hat der Ausgangsknoten keinen Nachbarn mehr in der entsprechenden Farbe und kann nun umgefärbt werden. Daraus leitet sich das folgende Vorgehen ab:
Wir beginnen mit der lokalen Verfeinerung. Wenn RecolorOrMove keine Änderung vornimmt, bestimmen wir eine Farbe und wenden RecolorOrMove auf alle Nachbarn des Knotens in dieser Farbe an. Wenn dies bei allen diesen Knoten gelingt, ist die gewählte Farbe für den aktuellen Knoten jetzt zulässig. Sonst wird bei den entsprechenden Knoten rekursiv mit dem gleichen Verfahren weiter gesucht.
Bei dieser Strategie muß allerdings bei den rekursiven Versuchen die Farbzahl weiter eingeschränkt werden. Die Farben, die bereits ausgewählt worden sind, dürfen nicht mehr für Knoten benutzt werden. Wenn also z.B. alle Nachbarn mit Farbe 2 eines Knotens mit Farbe 1 untersucht werden, darf natürlich auch Farbe 1 nicht weiter benutzt werden, schließlich ist das eigentliche Ziel, ohne Farbe 1 auszukommen.

Algorithmus 5 beschreibt dieses Vorgehen noch einmal ausführlich. Er ist in zwei Prozeduren geteilt, den Hauptteil RefineRecursive und die rekursive Suche RefineVertex. Der Beginn von RefineRecursive ist eine Kopie von Algorithmus 4, mit dem einzigen Unterschied, daß RefineVertex statt RecolorOrMove aufgerufen wird.
RefineVertex benötigt wie RecolorOrMove den aktuellen Knoten und das Repräsentantensystem $\mathcal{C}$ als Parameter. Zusätzlich wird die Menge F übergeben, in der festgehalten wird, welche Farben nicht verwendet werden dürfen.
Zuerst wird versucht, ob RecolorOrMove bereits zu einer Verbesserung führt, wobei auch dort keine Farbe aus F verwendet werden darf. Ansonsten wird die aktuelle Farbe verboten, damit keine weiteren Knoten diese Farbe erhalten. Anschließend wird die Farbe mit der höchsten Nummer gewählt, die nicht in F enthalten ist. Dann ruft sich RefineVertex rekursiv für alle Nachbarn des Knotens auf, die diese Farbe

Algorithmus 5 Rekursive Verfeinerung

```
 1: procedure RefineRecursive(G, C)
 2:     count = 0;
 3:     while count < MaxIter do
 4:         for i = c_max, ..., 1 do
 5:             for all v ∈ C_i do
 6:                 RefineVertex(v, i, ∅, C)
 7:             end for
 8:             if C_i = ∅ then
 9:                 vertausche Farben i und c_max
10:                 lösche C_{c_max}
11:                 c_max = c_max − 1
12:                 Sortiere C so daß wieder |C_1| ≥ ··· ≥ |C_{c_max}|
13:                 count = 0
14:                 goto 3
15:             end if
16:         end for
17:         count = count + 1
18:         Sortiere C so daß wieder |C_1| ≥ ··· ≥ |C_{c_max}|
19:     end while
20: end procedure

21: function RefineVertex(v, i, F, C)
22:     if RecolorOrMove(v, c_max, C, F) then
23:         return true
24:     end if
25:     F = F ∪ i
26:     j = max {i : i ∉ F}
27:     for all w ∈ δ(v) ∪ C_j do
28:         if not RefineVertex(w, j, F, C) then
29:             return false
30:         end if
31:     end for
32:     c(v) = j
33:     C_j = C_j ∪ v, C_i = C_i \ v
34:     F = F \ i
35:     return true
36: end function
```

haben.

4.3.4 Problemfälle

Die beiden vorgestellten Heuristiken verändern die Färbung nur, wenn eine lokale Verbesserung möglich ist. Die Idee ist, eher in einem anderen Block nach vermutlich besseren Knoten zu suchen als an einer bestimmten Stelle des Graphen viel Zeit zu investieren. Folglich werden bei der Suche alle Wahlen übergangen, bei denen sich die Farbe des Repräsentanten nicht ändert. Dies könnte in manchen Graphen zu nicht optimalen Ergebnissen führen. Abbildung 4.4 zeigt eine Konstellation, in der erst ein solcher Zug zur optimalen Wahl führt. Die Einträge in den Knoten sind entweder die Farbe oder "-" für "nicht gewählt". Beide in diesem Abschnitt vorgestellten Verfahren könnten hier keine Verbesserung finden, da alle Knoten im rechten Block mit dem Repräsentanten des linken Blockes verbunden sind und umgekehrt. Somit kann in keinem der beiden Blöcke mit den von uns zugelassenen Anpassungen erreicht werden, daß beide Repräsentanten dieselbe Farbe erhalten können. Erst die abgebildete Veränderung führt zum Ziel.

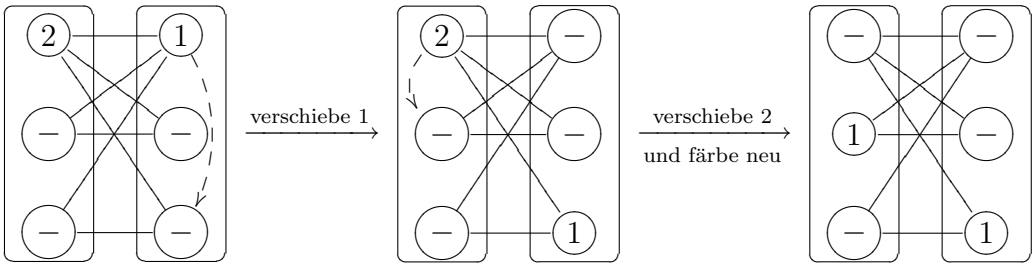


Abbildung 4.4: Problemfall, derzeit nicht abgedeckt

Der abgebildete Graph ist zwar nicht reduziert, allerdings läßt er sich mit zusätzlichen Blöcken so erweitern, daß kein Knoten mit der in Abschnitt 4.2 geschilderten Methode entfernt werden könnte. Abbildung 4.5 zeigt eine solche Erweiterung. Das Problem bleibt gleich, wenn als Repräsentant des zusätzlichen Blockes der Knoten "∘" gewählt wird.

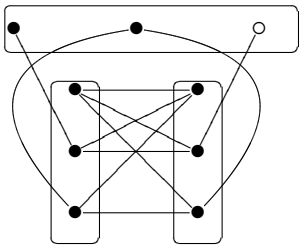


Abbildung 4.5: Erweiterung des Graphen aus Abbildung 4.4

Kapitel 5

Ergebnisse

In diesem Kapitel werden wir Resultate vorstellen, die wir mit Implementierungen der beiden Verfeinerungs-Heuristiken erzielt haben. Um die generelle Tauglichkeit der Verfahren zu testen, benutzen wir zufällig erzeugte Graphen, in denen jeder Block jeweils zehn Knoten enthält. Neben den Fragen nach der besten Wahl der Parameter, dem Einfluß der Startlösung und selbstverständlich der Laufzeit interessiert uns dabei auch das Verhalten in Abhängigkeit von der Graphenstruktur. Wir konzentrieren uns hier auf die Anzahl der Blöcke und die Dichte des Graphen, die angibt, mit welcher Wahrscheinlichkeit eine Kante zwischen zwei Knoten existiert. Insgesamt haben wir folgende Strukturparameter benutzt

Anzahl Blöcke: 50, 100, 150, 200, 250, 300, 400

Dichte: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%

Bei den unterschiedlichen Dichten haben wir jede Kante mit der angegebenen Wahrscheinlichkeit erzeugt, nicht unbedingt die angegebene Prozentzahl der möglichen Kanten.
Um den Einfluß der Startlösung auf die Ergebnisse zu untersuchen, benutzen wir für jeden Versuch 14 verschiedene Startlösungen, und zwar:

- die Lösungen, die von den drei Adaptionen des Greedy-Algorithmus aus Abschnitt 4.1.1 bestimmt werden
- die Lösungen, die nach [6] konstruiert werden, indem aus jedem Block ein Knoten mit möglichst kleinem Grad gewählt wird, und der von diesen Knoten induzierte Teilgraph mit dem Greedyalgorithmus gefärbt wird (ohne besondere Auswahlstrategie). Diese Strategie bezeichnen wir mit SmallestDegree.
- zehn zufällig ausgewählte Repräsentantensysteme, die mit dem Greedyalgorithmus gefärbt worden sind.

Die zufällig erzeugten Startlösungen sollen darüber Aufschluß geben, ob es sich lohnt, Zeit in die Berechnung einer möglichst guten Startlösung zu investieren.

Generell zeigt sich, daß unter den Startlösungen die DSatur-Adaption immer deutlich bessere Ergebnisse liefert als die übrigen Greedy-Varianten oder die zufällige Auswahl. Unter den übrigen Greedy-Varianten ist LargestFirst die beste, gefolgt von SmallestDegree und SmallestLast, wobei die Abstände dieser drei untereinander deutlich geringer sind als der Vorsprung von DSatur auf die zweitbeste Startlösung. Die genauen Ergebnisse sind im Anhang auf Seite 60 in Tabelle A.1 aufgeführt.

Die Laufzeitmessungen beziehen sich auf Testläufe auf einem Rechner mit DEC Alpha EV 6.7 Prozessor mit 667Mhz Takt, mit dem Betriebssystem Compaq Tru64 UNIX V5.1 (Rev. 732). Im Verhältnis zu den dort ermittelten Laufzeiten erreicht ein PC mit 2,4 Ghz etwa die vier- bis fünffache Leistung. Wir haben den `GNU g++` Compiler in Version 3.2.4 mit dem Optimierungsschalter `-O3` eingesetzt.

5.1 Lokale Verfeinerung

5.1.1 Lösungen

Die wichtigste Größe bei den Verfeinerungsheuristiken ist der Parameter MaxIter, mit dem gesteuert wird, nach wie vielen aufeinanderfolgenden Iterationen des Algorithmus ohne Verminderung der Farbzahl das Verfahren beendet wird. Zunächst wollen wir natürlich wissen, ob unsere Verfahren mindestens so gute Ergebnisse erreichen wie die beste Startlösung. In immerhin 35 der 63 Graphen war schon bei MaxIter = 10 selbst die schlechteste der vierzehn ermittelten Lösungen besser als die beste Startlösung. Die beste erzielte Lösung unterbot diesen Wert sogar in 52 Graphen. In nur 5 Graphen, alle mit Dichte 10% oder 20%, benutzte die beste Startlösung weniger Farben als die schlechteste mit unserem Verfahren berechnete Lösung. Insgesamt liegen die so erzielten Ergebnisse also zumindest im Bereich der Greedy-Algortithmen. Bei den hohen Dichten von 60% an aufwärts sind die ermittelten Ergebnisse zum Teil schon um mehrere Farben besser als die Startlösungen.
Eine Steigerung von MaxIter auf die Werte 50, 100 und 200 bringt kontinuierliche Verbesserungen der Ergebnisse. Vor allem die Qualität der schlechtesten errechneten Lösung steigt, diese sind ab etwa MaxIter = 200 bis auf wenige Ausnahmen besser als jede Startlösung. Außerdem nähern sich bei diesem Wert die beste und schlechteste Lösung meistens bis auf eine Farbe.
Weitere Steigerungen des Abbruchkriteriums bringen vor allem bei den mittleren Dichten und in den großen Graphen weitere Fortschritte. Dabei verbessern sich normalerweise einzelne Lösungen noch um eine Farbe, größere Erfolge wie beim Übergang von MaxIter von 10 zu 50 werden hier nicht mehr erzielt. In den Graphen mit Dichten oberhalb von 70% ändern sich die Lösungen kaum noch.
Dieses Verhalten illustrieren die Abbildungen 5.1, 5.2 und 5.3. Sie zeigen die Entwicklung der durchschnittlichen Farbzahl mit steigender Iterationszahl für die Graphen

mit Dichte 10%, 50% und 90%. Die Werte sind jeweils über die Ergebnisse mit allen 14 Startlösungen gemittelt. Hier wird noch einmal deutlich, daß die Qualität der Lösungen mit steigender Dichte der Graphen früher relativ stabil wird. Insbesondere bei Dichte 50% fällt auf, daß die Erhöhung von MaxIter auf 500 stellenweise noch Verbesserungen bringt.

Die Methode, mit der die Startlösungen bestimmt werden, scheint keinen erheblichen Einfluß auf die Ergebnisse zu haben. Die Anzahl benutzter Farben schwankt zwar bei unterschiedlichen Anfangsfärbungen, doch ist dieser Effekt weitgehend unabhängig von der Farbzahl der Startlösung. Natürlich kann die Farbzahl der ursprünglichen Färbung nicht überschritten werden. Dies führt dazu, daß die Ergebnisse, die auf der DSatur-Adaption basieren immer mindestens so gut wie diese beste Startlösung sind, was bei kleinen MaxIter-Werten durch die zufällig gewählten Repräsentanten nicht immer erreicht wird. Bei großen Werten von MaxIter oder mittleren bis hohen Dichten zeigt sich dieser Effekt nicht mehr, und alle Startlösungen führen in einigen Fällen zum besten und in anderen zum schlechtesten Ergebnis.

Der Einfluß der Graphenstruktur ist wie erwartet. Die Abbildungen 5.4 und 5.5 zeigen die Auswirkungen der Anzahl Blöcke und Dichte der Graphen auf die Farbzahl, wobei wir jeweils die gemittelten Werte mit MaxIter = 1000 verwenden. Erwartungsgemäß steigt die Zahl verwendeter Farben in etwa linear mit der Anzahl Blöcke im Graphen, wobei der Anstieg bei höherer Dichte stärker ist. Dies deckt sich mit der Erwartung, da in dichten Graphen die maximalen stabilen Mengen wegen der höheren Kantenzahl normalerweise weniger Knoten enthalten. Bei fester Größe eines Graphen steigt die Farbzahl mit wachsender Dichte, wobei hier ein angenähert exponentielles Verhalten beobachtet werden kann.
Detaillierte Daten zu den Ergebnissen sind in Tabelle A.2 auf Seite 62 aufgeführt.

5.1.2 Laufzeit

Die Laufzeiten mit unterschiedlichen Startlösungen zeigen teilweise große Schwankungen, insbesondere bei den Graphen mit vielen Blöcken. Während die meisten Läufe annähernd die gleiche Zeit benötigen, liegen einzelne Extrema zum Teil um den Faktor zwei auseinander. Dies scheint hauptsächlich daran zu liegen, daß die Fortschritte nach unterschiedlich vielen Iterationen erzielt werden. Wenn die Farbzahl immer erst nach fast MaxIter Iterationen sinkt, werden sehr viel mehr Durchläufe gemacht als wenn die Farbzahl mit wenigen Iterationen reduziert werden kann. Dies ist eine direkte Auswirkung der Strategie, auf eine bestimmte Anzahl erfolgloser Durchläufe zu warten anstatt eine feste Iterationszahl vorzugeben. Diese Technik ist jedoch aus dem selben Grund notwendig, um unabhängig von der gewählten Startlösung möglichst gleichbleibende Ergebnisse zu erreichen.
Wie schon bei den Ergebnissen zeigt sich kein eindeutiges Bild zu Gunsten eines bestimmten Typs Startlösung. In der Tendenz schneidet hier die DSatur-Startlösung allerdings meistens recht gut ab. Auch hier zeigt sich wieder der oben beschrie-

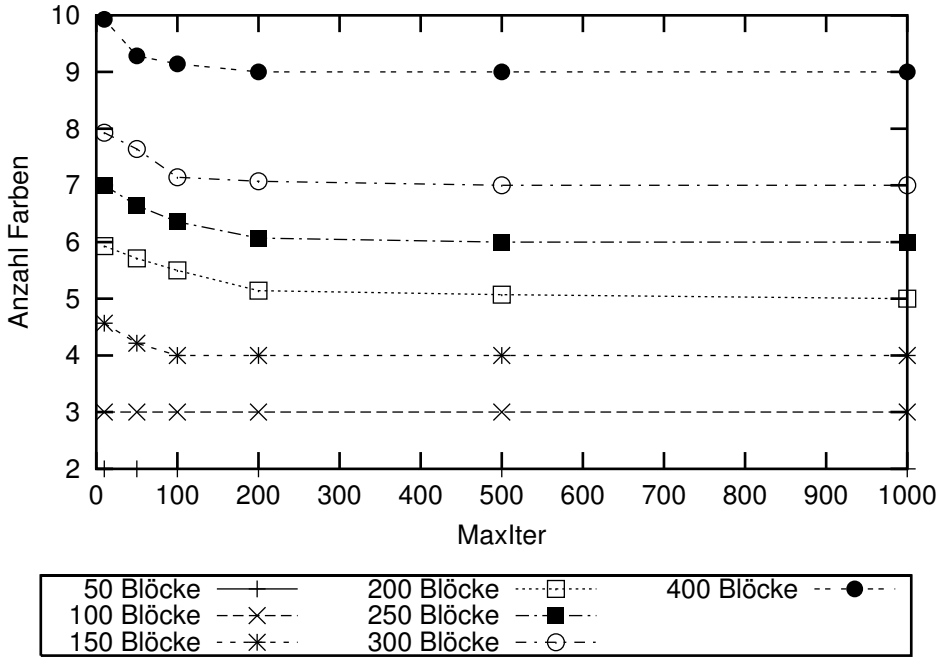


Abbildung 5.1: Entwicklung der Farbzahl bei 10% Dichte

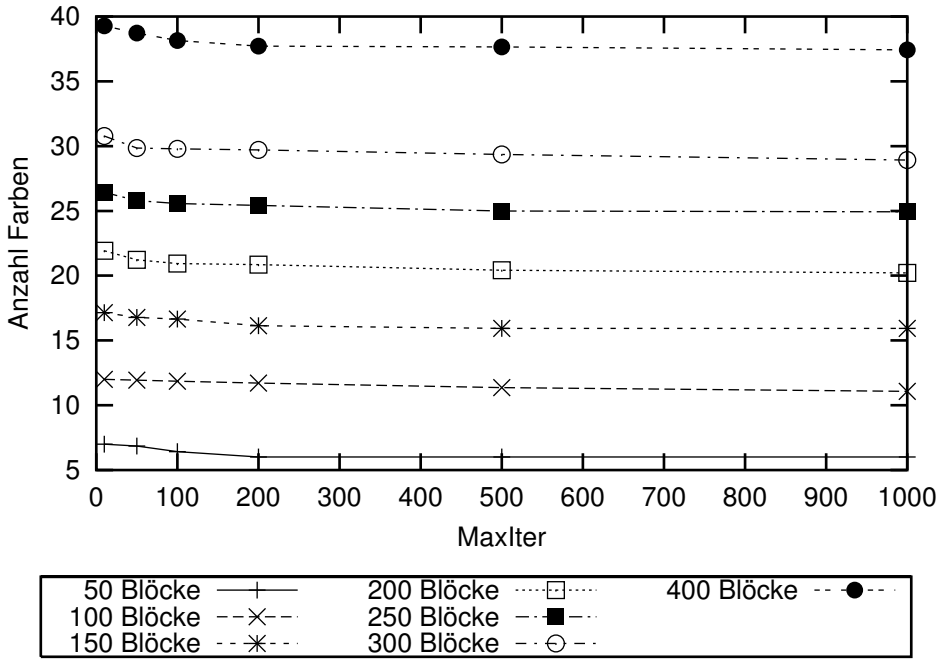


Abbildung 5.2: Entwicklung der Farbzahl bei 50% Dichte

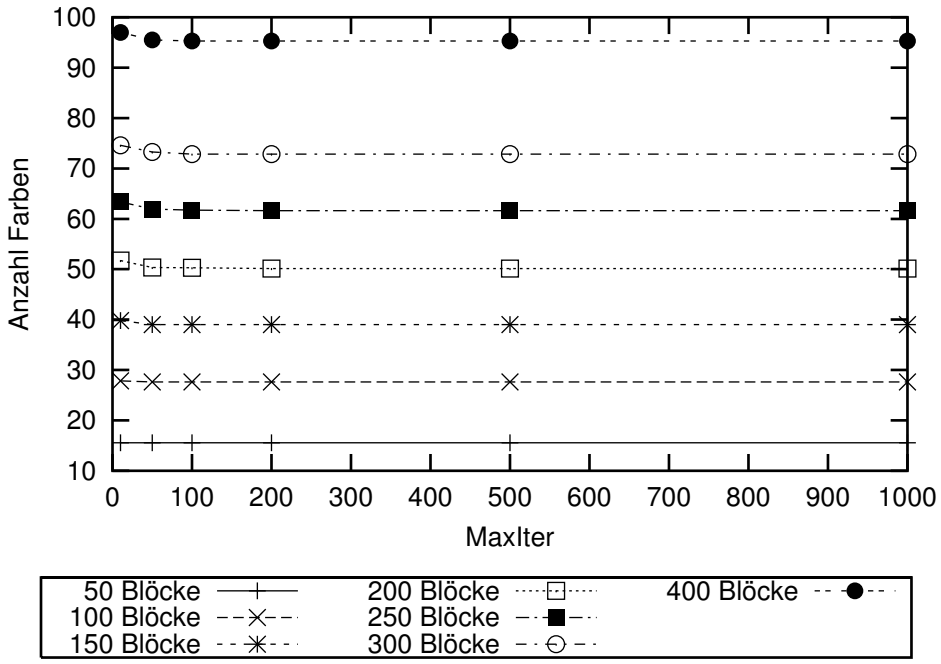


Abbildung 5.3: Entwicklung der Farbzahl bei 90% Dichte

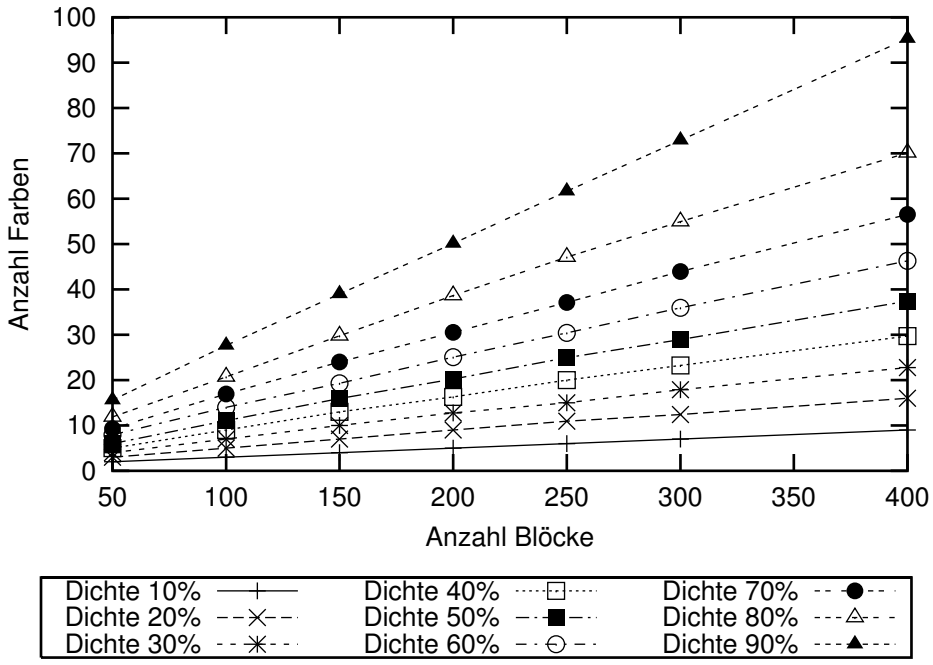


Abbildung 5.4: Die Farbzahl in Abhängigkeit von der Anzahl der Blöcke (MaxIter = 1000)

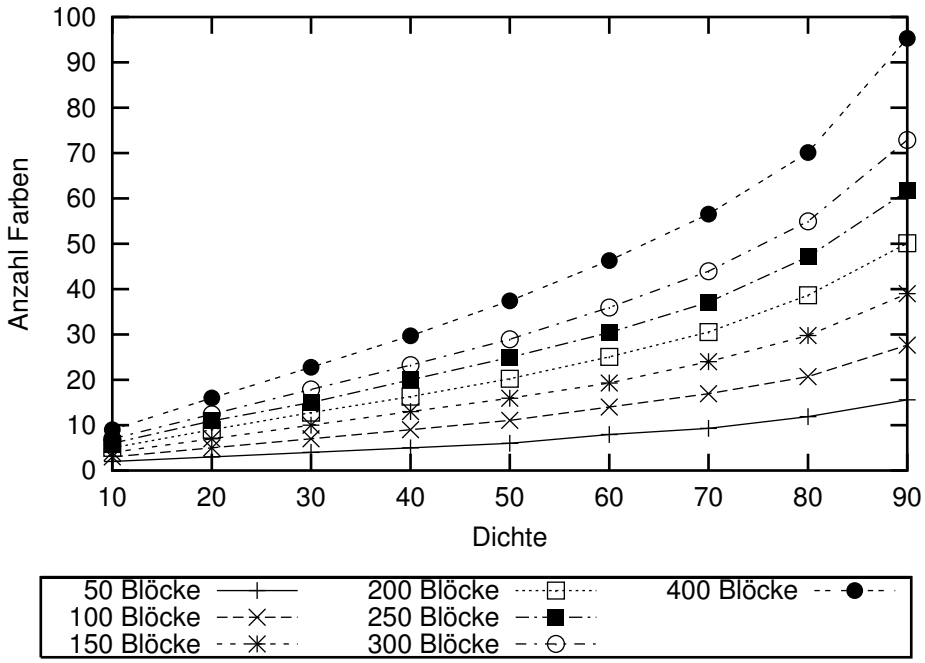


Abbildung 5.5: Einfluß der Dichte auf die Farbzahl (MaxIter = 1000)

bene Effekt. Bei guten Startlösungen sind weniger erfolgreiche Iterationen bis zu einer gewissen Farbzahl erforderlich, wodurch die Laufzeiten zumindest in den meisten Fällen recht gut sind. Allerdings werden ähnliche oder leicht bessere Laufzeiten auch von manchen zufällig gewählten Startlösungen erreicht.

Die gemittelte Laufzeit wächst wie erwartet in etwa linear mit steigenden Werten von MaxIter. Die einzelnen Iterationen der Heuristik können in Linearzeit ausgeführt werden. Im schlimmsten Fall wird für jeden Knoten RecolorOrMove aufgerufen. In der Summe ergibt sich damit nach Satz 4.8 die lineare Laufzeit. Die Abbildungen 5.6 und 5.7 zeigen diese Entwicklung für 50% und 90% Dichte.

Bezogen auf eine feste Anzahl Blöcke steigt die Laufzeit mit größer werdender Dichte stark an, wie man an den Abbildungen 5.8 und 5.9 sieht. Auffällig ist, daß die Laufzeit bei 400 Knoten bei Dichte 90% wieder sinkt. Dieser Effekt ist auch bei den übrigen Werten von MaxIter zu beobachten. Anscheinend ist dieser Graph so dicht, daß nach frühen Verbesserungen schnell die zulässige Anzahl erfolgloser Iterationen erreicht wird. Bei den übrigen Graphen zeigt sich dieses Verhalten nicht, hier steigt die Laufzeit stark mit Dichte oder Anzahl Blöcke an. Abbildung 5.10 zeigt das Laufzeitverhalten mit wachsender Anzahl Blöcke im Graphen. Auch hier sieht man den im Vergleich mit 70% oder 80% deutlich geringeren Anstieg des Zeitbedarfs bei 400 Knoten bei 90% Dichte.
Tabelle A.3 (Seite 64) zeigt noch einmal genaue Daten zu den Graphiken.

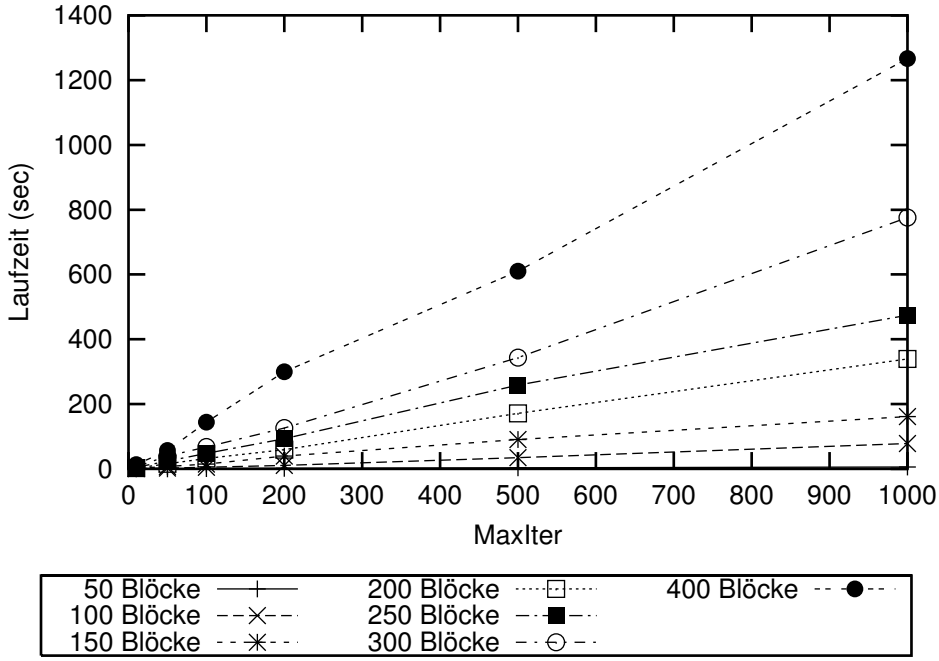


Abbildung 5.6: Entwicklung der Laufzeit bei 50% Dichte

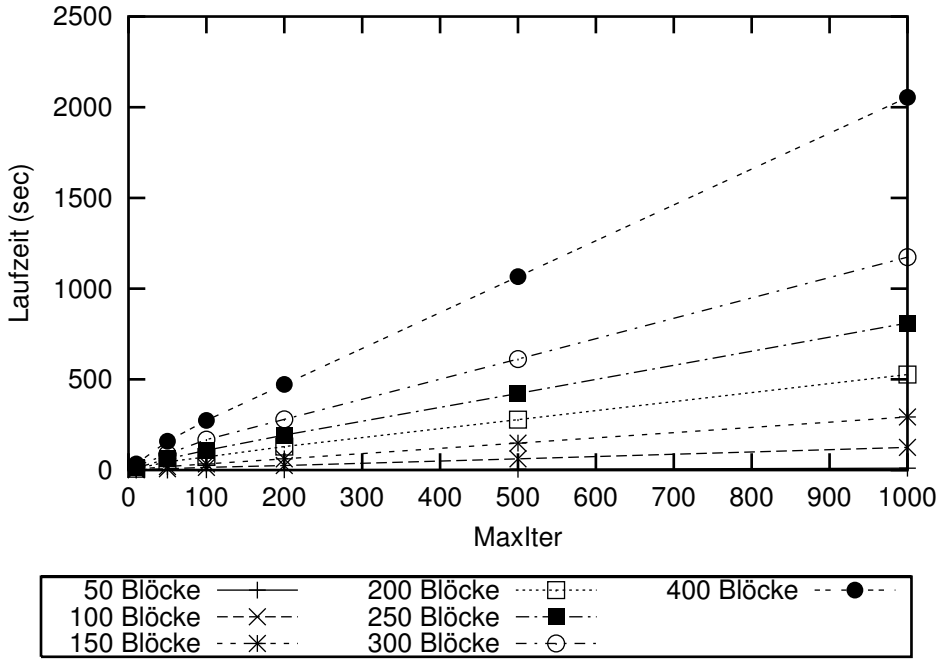


Abbildung 5.7: Entwicklung der Laufzeit bei 90% Dichte

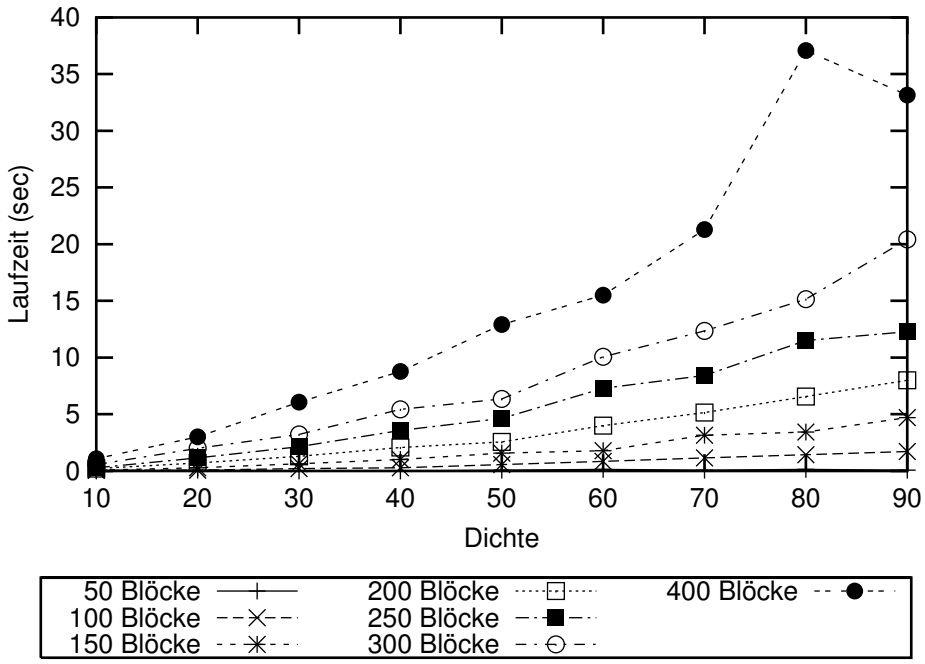


Abbildung 5.8: Laufzeiten abhängig von der Dichte (MaxIter = 10)

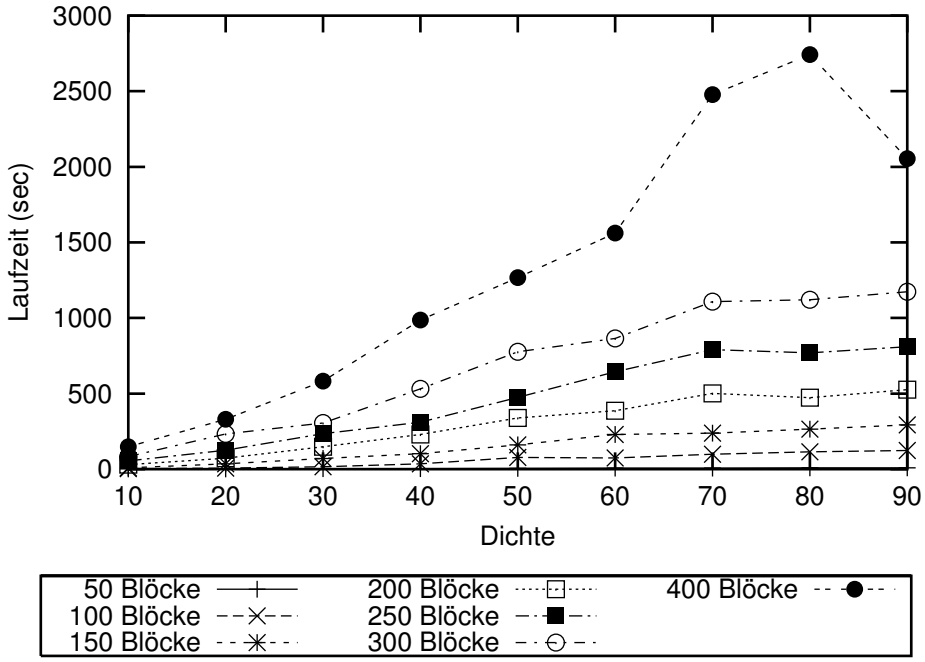


Abbildung 5.9: Laufzeiten abhängig von der Dichte (MaxIter = 1000)

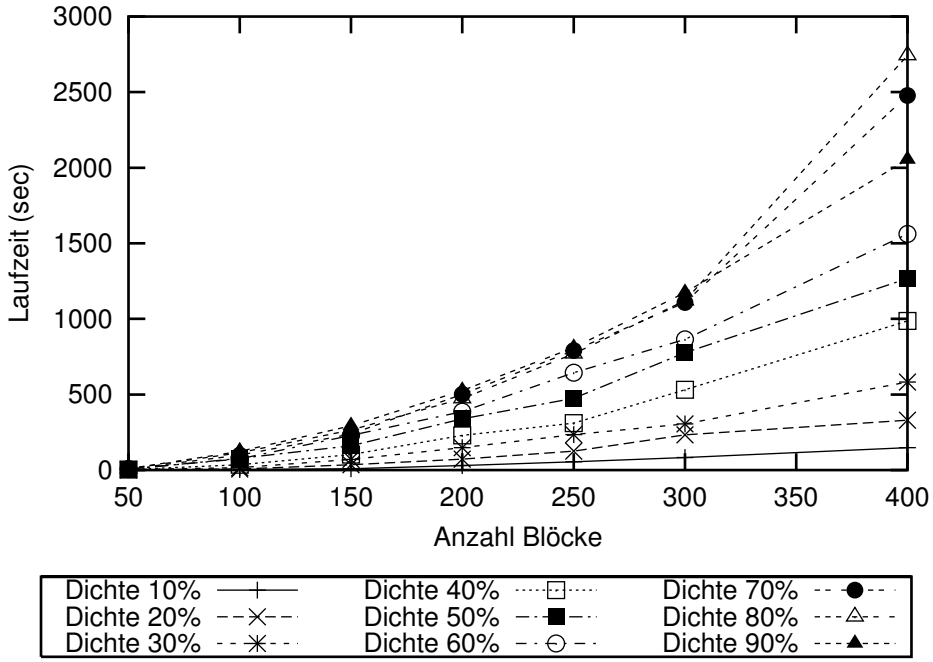


Abbildung 5.10: Laufzeiten bei unterschiedlicher Anzahl Blöcke (MaxIter = 1000)

5.2 Rekursive Verfeinerung

Die Messwerte für die rekursive Verfeinerung beziehen sich auf dieselben Graphen wie im vorangegangenen Abschnitt. Allerdings haben wir aus Zeitgründen auf die Auswertung der beiden größten Graphen verzichtet und präsentieren hier nur die Ergebnisse für die Instanzen bis einschließlich 250 Blöcke. Die Werte für das Abbruchkriterium MaxIter müssen selbstverständlich an die veränderte Vorgehensweise des Verfahrens angepaßt werden. Bei dem rekursiven Verfahren ist die Erfolgsquote einer Iteration erwartungsgemäß höher, da durch die erweiterte und zielstrebigere Suche mit größerer Wahrscheinlichkeit eine Farbe eliminiert wird. Außerdem werden bei den rekursiven Aufrufen bereits weitere Veränderungen am Repräsentantensystem vorgenommen. Insgesamt erwarten wir daher, daß kleinere Werte für MaxIter ausreichen. Hier beschränken wir uns auf die Werte 10, 20, 30 50 und 100. Wie schon im vorangegangenen Abschnitt sind Teile der Ergebnisse im Anhang in den Tabellen A.4 und A.5 ab Seite 66 aufgeführt.

5.2.1 Lösungen

Im Vergleich mit der besten Startlösung zeigt sich, daß hier bereits ein MaxIter-Wert von 10 zu relativ guten Ergebnissen führt. Schon bei diesem Wert unterschreitet die schlechteste erzielte Lösung die beste Startlösung in immerhin 30 der 45 Graphen. Die beste Lösung erreicht dies sogar in 39 Graphen. Wie schon bei der lokalen Verfeinerung werden die besten Resultate bei den Graphen mit hoher Dichte erreicht, während bei den niedrigeren Dichten länger gerechnet werden muß, bis auch die schlechteste Lösung mindestens so gut wie die beste Startlösung ist. In den Graphen mit 50 und 200 Knoten bei Dichte 10% wird die beste Startlösung von jeweils einer Lösung zumindest mit den untersuchten Einstellungen nicht erreicht.
Insgesamt zeigt sich bei der Steigerung des Wertes MaxIter eine leichte, kontinuierliche Verbesserung der Werte. Meistens verbessern sich etwa zwei bis vier der vierzehn Lösungen beim Übergang zu einem größeren Wert. Dies gilt insbesondere auch bei einer Steigerung auf MaxIter = 100, so daß in diesem Bereich noch keine endgültige Stabilisierung der Ergebnisse eintritt. Auch die erzielten besten oder schlechtesten Lösungen können noch schwanken. Diese Tendenz zeigen die Abbildungen 5.11, 5.12 und 5.13. Stichproben zeigen jedoch, daß mit höheren MaxIter-Werten keine signifikanten Verbesserungen mehr zu erzielen sind.
Abgesehen davon, daß auch hier wieder die Farbzahl der jeweiligen Startlösung nicht überschritten werden kann, zeigt sich auch hier kein besonderer Vorteil einer speziellen Art der Startlösung. Wie schon beim lokalen Verfahren werden die Extrema der Farbzahlen bei allen Arten von Startlösungen angenommen.

Der Einfluß der Graphenstruktur hat sich durch die Wahl eines anderen Verfahrens nicht geändert, so daß wir diese Analyse hier nicht noch einmal durchführen.

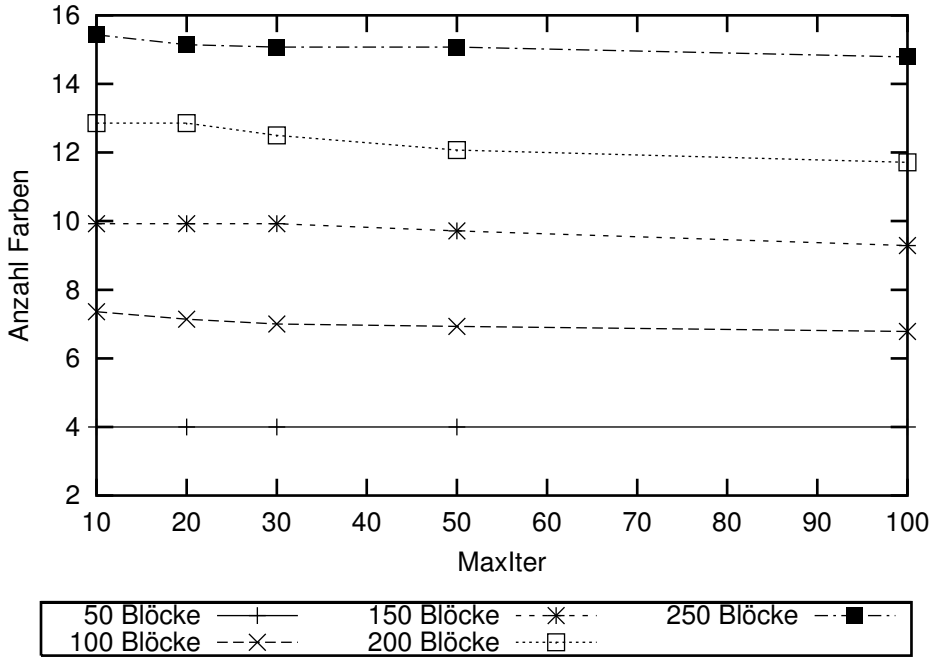


Abbildung 5.11: Entwicklung der Farbzahl bei 30% Dichte

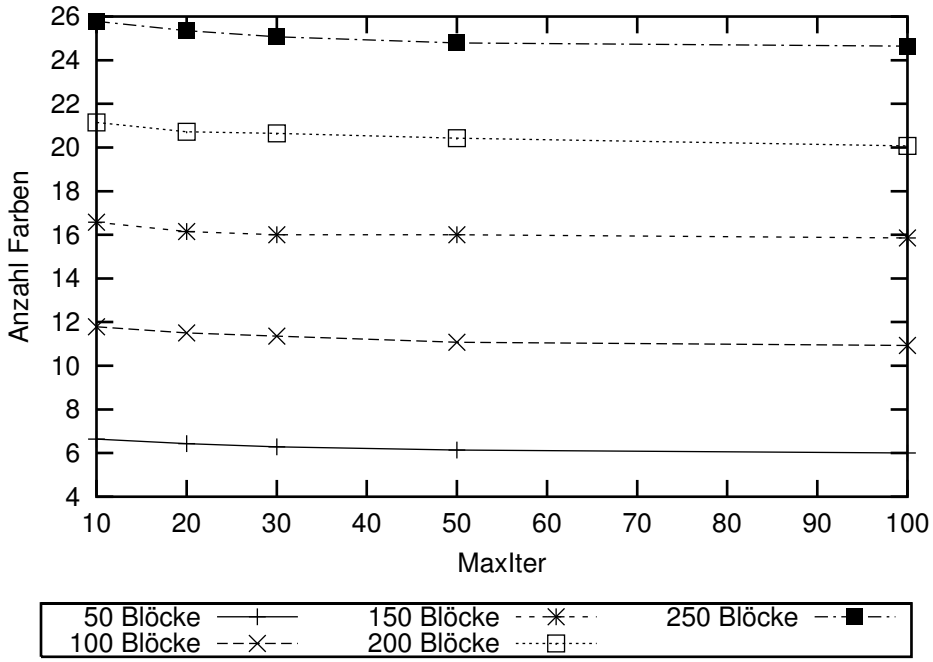


Abbildung 5.12: Entwicklung der Farbzahl bei 50% Dichte

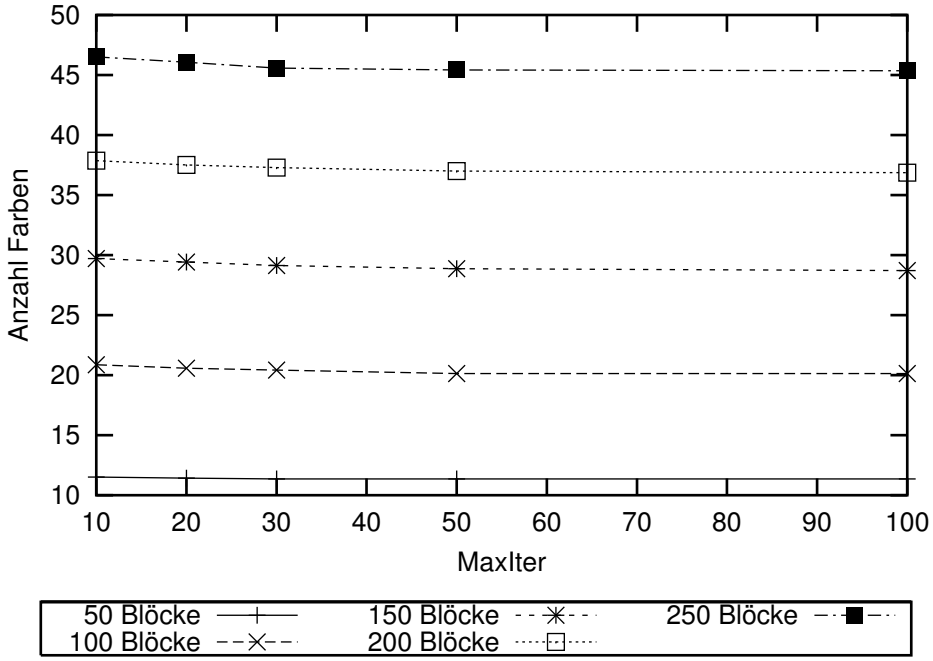


Abbildung 5.13: Entwicklung der Farbzahl bei 80% Dichte

5.2.2 Laufzeit

Die enormen Schwankungen der Laufzeiten bei unterschiedlichen Startlösungen treten auch bei diesem Verfahren auf, sogar noch stärker. Vereinzelt kann eine Lösung dreimal so lange benötigen wie eine andere. Zusätzlich zu dem bereits beschriebenen Einfluß der Anzahl erfolgloser Iterationen kommt bei diesem Verfahren die Rekursionstiefe hinzu. Auch in erfolgreichen Iterationen können sehr wenige oder auch sehr viele rekursive Schritte durchgeführt worden sein, was die Schwankungen mit hervorruft.
Ein unmittelbarer Einfluß der Methode, mit der die Startlösungen bestimmt worden sind, kann wieder nicht beobachtet werden. Allerdings zeigt sich, daß mit den beiden besten Startlösungen DSatur und LargestFirst fast immer Ergebnisse in der unteren Hälfte der gemessenen Spanne erzielt werden. Dies ist vermutlich ebenso wie bei der lokalen Verfeinerung auf die insgesamt niedrigere Zahl erfolgreicher Iterationen zurückzuführen. Die Laufzeiten mit zufällig gewählten Startlösungen sind über den gesamten benutzten Bereich verteilt.

Abgesehen von den Schwankungen steigt die gemittelte Laufzeit mit zunehmenden MaxIter-Werten massiv an. In den Abbildungen 5.14 und 5.15 zeigen wir die Entwicklung in den Graphen mit 30% und 80% Dichte. Hier ist vor allem auffällig, daß die Zunahme beim Übergang von 50 zu 100 als Abbruchkriterium langsamer verläuft als bei den kleineren Werten von MaxIter. Statt sich wie beim Übergang von 20 zu 50 ungefähr zu verdreifachen, verdoppelt sich die Laufzeit bei diesem

letzten Übergang. Dies liegt wohl daran, daß in den meisten Fällen nur noch einmal eine weitere Farbe eingespart werden kann und danach das Abbruchkriterium erfüllt wird.

Der Einfluß der Graphenstruktur ist ähnlich wie beim lokalen Verfahren, nur daß die entsprechenden Steigerungen stärker ausfallen. In den Abbildung 5.16 und 5.17 wird das entsprechende Verhalten für MaxIter = 100 dargestellt. Für die anderen Werte dieses Parameters ergeben sich ähnliche Bilder, nur mit anderen absoluten Zahlen.

5.3 Graphen aus dem Primer-Design

Zusätzlich zu den beschriebenen künstlich erzeugten Graphen haben wir die Verfahren auch mit drei Graphen getestet, die beim Primer-Design entstehen. Diese Graphen sind relativ dünn, es existieren etwa 20% der möglichen Kanten. Typisch enthalten sie etwa 50 Blöcke mit 40 bis 60 Knoten. Im Gegensatz zu den zufällig generierten Graphen zeigt das Preprocessing aus Abschnitt 4.2 hier eine enorme Wirkung und entfernt etwa ein Drittel der Knoten. Diese Reduktion ist möglich, da die längeren Primer die kürzeren als Teilstrang enthalten und somit auch zu allen Primern inkompatibel sind, zu denen diese Teilstränge es auch sind.
Da diese Graphen wenige Kanten enthalten, bestimmt der DSatur-Greedy oft schon eine sehr gute Startlösung. In zwei der drei Graphen konnte diese Lösung nicht unterboten werden. Dabei mußten MaxIter nur auf 50 (lokal) beziehungsweise 20 (rekursiv) gewählt werden, um diese Farbzahl immer zu erreichen. Im dritten Graphen konnte die beste Startlösung mit rekursiver Verfeinerung um eine Farbe verbessert werden, was ab MaxIter = 50 auch mit allen Startlösungen gelang. Der lokalen Suchstrategie gelang dies selbst mit dem höchsten Wert für MaxIter nicht immer, vor allem mit den guten Startlösungen.
Die Laufzeiten waren bei diesen Graphen erfreulich, keine Rechnung benötigte mehr als zwei Minuten. Bei diesen Größen spielen die Nachteile der rekursiven Technik noch keine entscheidende Rolle.

5.4 Fazit

Beide Verfahren bestimmen auf der Testmenge gute Lösungen. Bei den dünnen Graphen ist das lokale Verfahren zuverlässiger und liefert bereits bei MaxIter = 200 meistens mit allen Startlösungen dieselbe Farbzahl. Im Gegensatz dazu schafft es das rekursive Verfahren bei diesen Graphen nicht immer, die beste Startlösung zu erreichen. Das rekursive Verfahren ist dagegen bei den dichten Graphen (Dichte über 50%) überlegen, ab MaxIter = 20 ist der Durchschnitt der Farbzahl um 0.3 bis 0.5 niedriger als bei den lokalen Verfahren. Mit steigenden Werten von MaxIter wächst

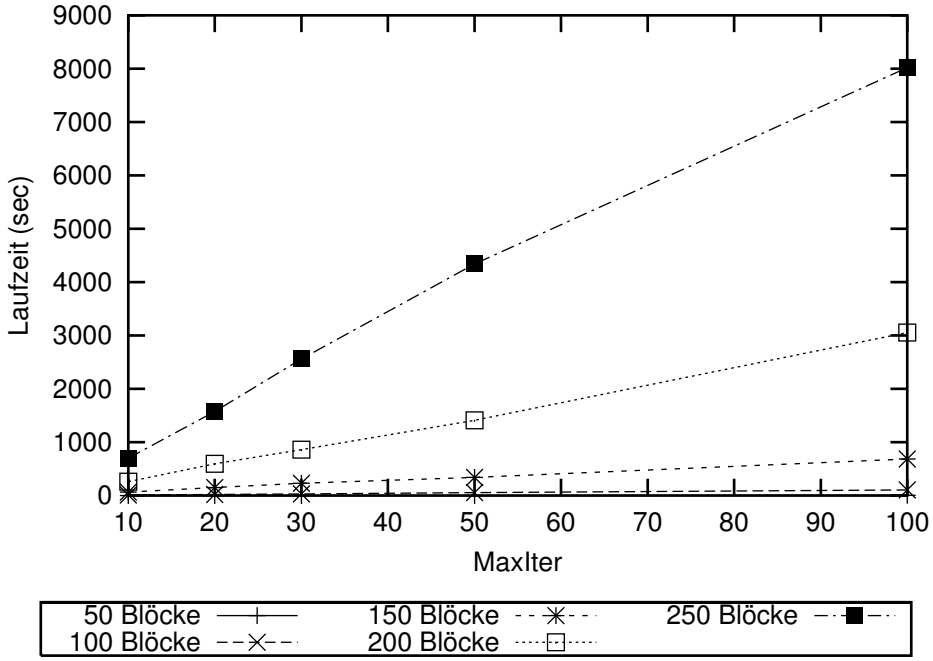


Abbildung 5.14: Entwicklung der Laufzeit bei 50% Dichte

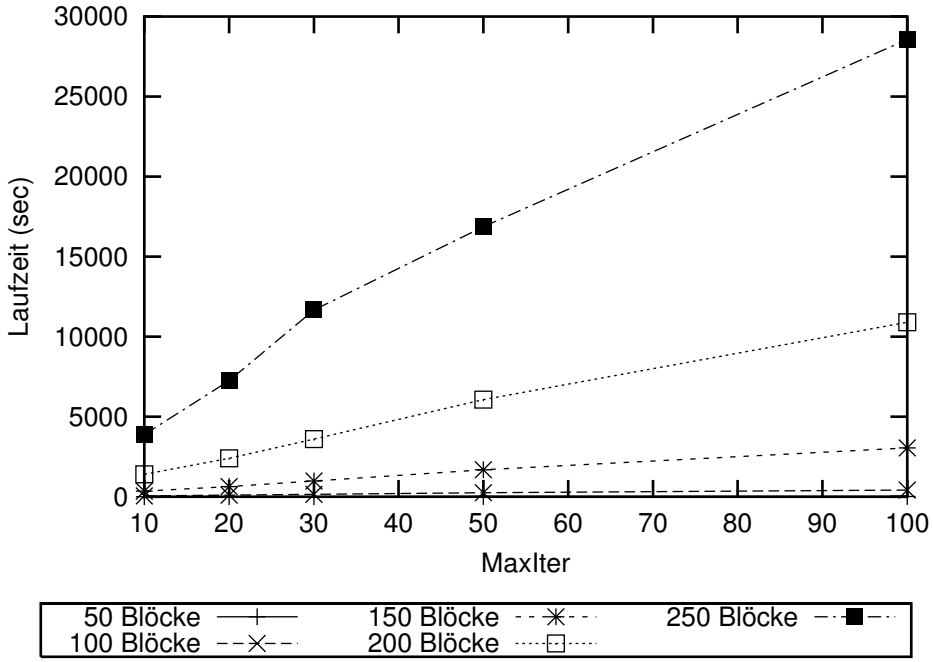


Abbildung 5.15: Entwicklung der Laufzeit bei 80% Dichte

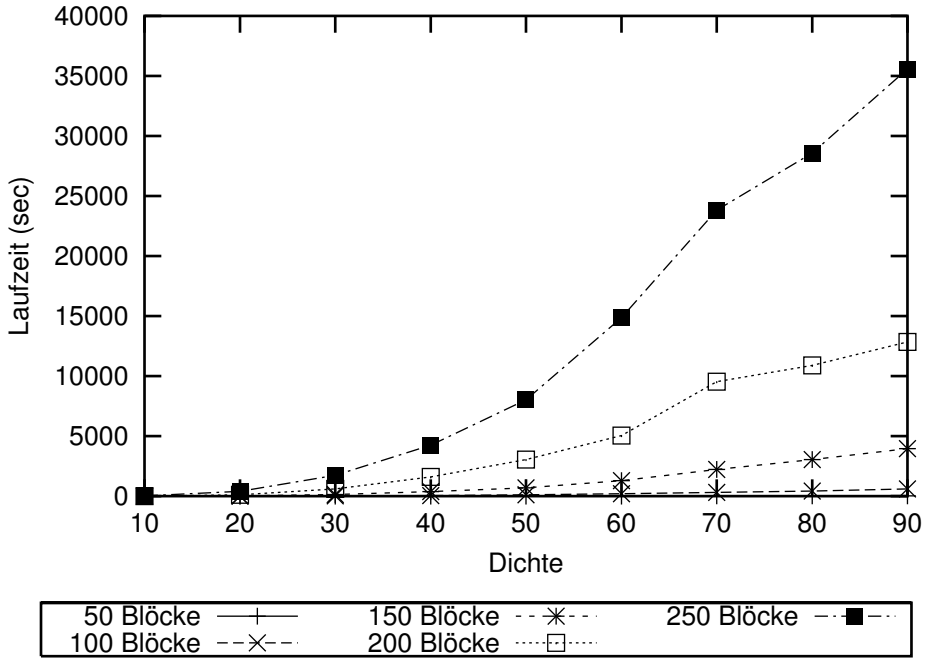


Abbildung 5.16: Entwicklung der Laufzeit abhängig von der Dichte (MaxIter=100)

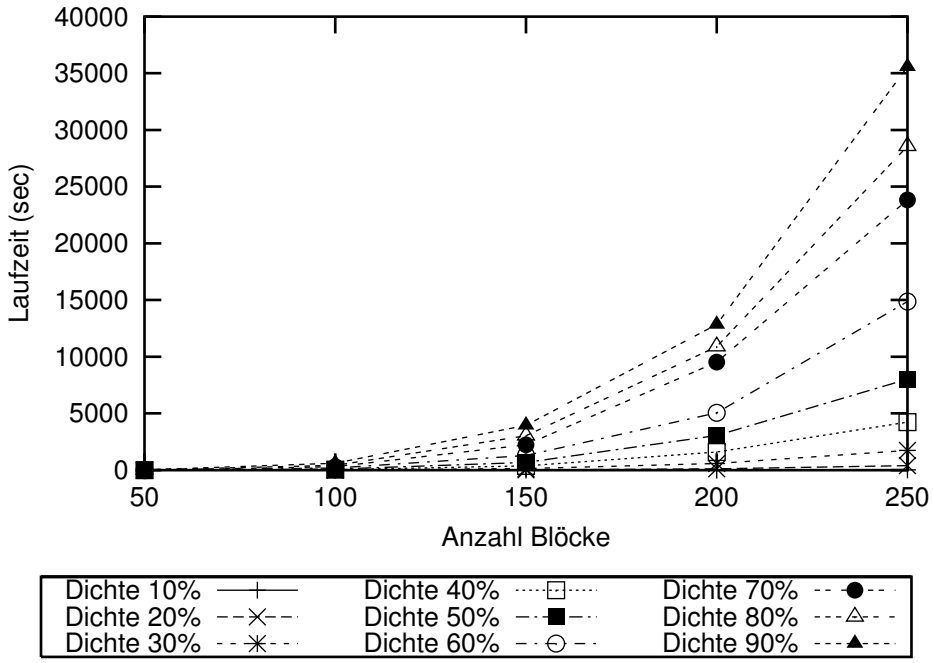


Abbildung 5.17: Entwicklung der Laufzeit abhängig von der Anzahl Blöcke (MaxIter=100)

dieser Vorsprung weiter, besonders bei 250 Blöcken kann der Vorsprung auf fast eine Farbe wachsen.
Dabei sollte man nicht vergessen, daß die Laufzeiten der rekursiven Verfeinerung deutlich höher sind als die des lokalen Verfahrens. Selbst wenn nach 20 erfolglosen Iterationen abgebrochen wird, läuft das Verfahren bei dichten Graphen schon bis zu dreimal so lange. Der enorme Mehrbedarf an Laufzeit bei späterem Abbruch steigert diesen Faktor auf bis zu 10 bei dem größten von uns untersuchten Wert von MaxIter (100). Hier muß allgemein also ein Kompromiss aus Laufzeit und Qualität der Ergebnisse gefunden werden. Insgesamt legt das Laufzeitverhalten nahe, mit guten Startlösungen zu beginnen und so den Zeitaufwand soweit möglich gering zu halten.
Damit ergibt sich kein eindeutig bestes Verfahren, sondern es muß nach Beschaffenheit der entsprechenden Graphen und der maximalen zulässigen Laufzeit entschieden werden, welche Strategie benutzt werden soll. Als Faustregel ist außer bei sehr dünnen Graphen die rekursive Verfeinerung jedoch vorzuziehen, wenn der erhöhte Zeitaufwand unproblematisch ist.
Genauso muß die Wahl der Parameter problemspezifisch erfolgen, eine allgemeingültige Regel gibt es dazu nicht. Hier muß der geeignete Kompromiss aus Laufzeit und Qualität gefunden werden. Es scheint jedoch so, daß oberhalb der hier beschriebenen Werte nur noch marginale Verbesserungen zu erwarten sind, so daß die hier beschriebenen Werte als Indikator dienen können. Beim lokalen Verfahren spricht wenig dagegen, MaxIter auf 1000 zu setzen, da die Laufzeit für die meisten Anwendungen akzeptabel sein dürfte. Für das rekursive Verfahren sollte dieser Wert nicht unter 30 gewählt werden, besser noch höher. Die genaue Auswahl muß hier auf die Anforderungen an das Laufzeitverhalten abgestimt werden.

Ausblick

Im Bereich der Heuristiken sind noch Probleme offen. Weitere heuristische Techniken wie Simulated Annealing können auf ihre Tauglichkeit für dieses Problem überprüft werden. Auch Modifikationen der vorgestellen Verfahren sind denkbar, um die Qualität der Lösungen weiter zu verbessern oder den Zeitbedarf zu verringern.

Vor allem die theoretische Analyse des Problems befindet sich noch im Anfangsstadium. Für den allgemeinen Fall sind wie gezeigt effiziente Lösungsverfahren nicht zu erwarten, es sei denn $P = NP$. Es bleibt jedoch die Frage offen, ob das Problem zumindest in speziellen Graphenklassen in polynomieller Zeit gelöst werden kann. Weiterhin muß noch erforscht werden, ob χ_P wenigstens in Spezialfällen mit einer gewissen Qualität approximiert werden kann. Insbesondere wären Verfahren hilfreich, die eine Färbung mit höchstens $c\ \chi_P$ Farben mit einem konstanten Faktor c bestimmen.

Anhang A

Ausgesuchte Daten

In diesem Kapitel geben wir zusätzlich zur Auswertung in Kapitel 5 Teile der Messwerte an. Um die Darstellung dabei übersichtlich zu halten, beschränken wir uns auf im Text besonders erwähnte Parameter und Daten, die die Entwicklung der Ergebnisse verdeutlichen.

Tabelle A.1: Farbzahlen der Startlösungen. SD=SmallestDegree, SL=SmallestLast, LF=LargestFirst, DS=DSatur. Für die zufällig gewählten Startlösungen geben wir Minimum, Durchschnitt und Maximum an.

		Greedy-Adaptionen				Zufällige Wahl		
#Blöcke	Dichte	SD	SL	LF	DS	min	av	max
50	10%	4	4	3	2	5	5	5
50	20%	6	6	6	3	7	7	7
50	30%	8	9	7	5	8	8,8	9
50	40%	9	11	7	6	10	10,8	12
50	50%	11	12	12	7	12	12,8	13
50	60%	12	12	13	9	14	15	17
50	70%	16	17	15	11	17	17,4	18
50	80%	20	21	19	14	20	21,3	22
50	90%	25	26	24	17	24	26,4	29
100	10%	6	7	6	3	7	7,6	8
100	20%	9	10	8	5	10	10,4	11
100	30%	12	13	12	8	12	13,8	15
100	40%	16	16	16	10	16	17,6	19
100	50%	19	19	18	13	20	21,3	23
100	60%	22	24	23	16	24	25,5	27
100	70%	28	30	27	19	29	30,8	32
100	80%	32	34	30	24	35	36,5	37
100	90%	41	46	40	30	45	47	50
150	10%	9	8	7	4	8	9	10
150	20%	12	13	12	8	13	14,2	16
150	30%	17	17	16	11	17	18,6	19
150	40%	22	23	21	14	23	24,2	26
150	50%	25	28	26	18	27	28,7	30
150	60%	33	34	32	22	33	35,2	37
150	70%	40	39	39	27	40	41,6	43
150	80%	49	49	48	34	48	50,7	53
150	90%	60	64	57	47	63	66	70

Fortsetzung auf der nächsten Seite

Fortsetzung von Tabelle A.1								
		Greedy-Adaptionen				Zufällige Wahl		
#Blöcke	Dichte	SD	SL	LF	DS	min	av	max
200	10%	10	10	9	5	10	10,7	11
200	20%	15	16	14	10	15	16,7	18
200	30%	21	22	20	14	22	22,5	24
200	40%	27	27	26	19	28	29,2	30
200	50%	33	34	32	23	35	36,1	39
200	60%	40	41	39	29	42	43,8	46
200	70%	51	50	46	35	51	52,1	54
200	80%	61	59	59	46	63	64,7	66
200	90%	77	80	75	60	78	81,9	86
250	10%	11	11	10	7	12	12,9	14
250	20%	17	20	17	11	19	19,8	21
250	30%	25	25	23	16	26	27,1	29
250	40%	32	32	31	23	33	34,3	35
250	50%	41	40	38	28	41	41,9	43
250	60%	49	52	46	35	51	51,7	53
250	70%	62	61	58	43	59	62,3	64
250	80%	74	76	72	54	73	76,2	78
250	90%	93	97	89	72	94	98,5	103
300	10%	12	13	11	8	13	13,7	14
300	20%	21	21	20	14	21	22	23
300	30%	28	30	28	19	29	30,4	32
300	40%	36	36	35	25	37	39,4	41
300	50%	45	47	45	33	46	48,9	51
300	60%	53	56	55	41	58	59,6	61
300	70%	67	67	67	53	71	72,8	75
300	80%	84	86	83	63	87	89,6	94
300	90%	104	112	107	84	111	115	117
400	10%	15	16	14	9	16	17,1	18
400	20%	26	27	24	17	27	27,2	28
400	30%	36	35	33	24	36	37,7	39
400	40%	48	48	45	33	47	48,8	50
400	50%	57	60	56	42	59	61,1	62
400	60%	70	73	67	51	73	75,4	77
400	70%	86	89	84	63	89	91,5	94
400	80%	107	109	103	81	109	112,6	115
400	90%	144	139	135	105	144	147,2	152

Tabelle A.2: Ergebnisse der lokalen Verfeinerung bei MaxIter $= 10, 200, 1000$. Gegeben sind die Anzahl Blöcke, die Dichte, die Farbzahl der besten Startlösung (DSatur-Adaption) sowie jeweils die minimale, durchschnittliche und maximale Farbzahl der Lösungen

			MaxIter = 10			MaxIter = 200			MaxIter = 1000		
#Blöcke	Dichte	DS	min	av	max	min	av	max	min	av	max
50	10%	2	2	2,00	2	2	2,00	2	2	2,00	2
50	20%	3	3	3,00	3	3	3,00	3	3	3,00	3
50	30%	5	4	4,36	5	4	4,00	4	4	4,00	4
50	40%	6	5	5,86	6	5	5,00	5	5	5,00	5
50	50%	7	7	7,00	7	6	6,00	6	6	6,00	6
50	60%	9	8	8,00	8	8	8,00	8	7	7,93	8
50	70%	11	9	9,86	10	9	9,29	10	9	9,29	10
50	80%	14	11	11,93	13	11	11,86	12	11	11,86	12
50	90%	17	15	15,57	16	15	15,57	16	15	15,57	16
100	10%	3	3	3,00	3	3	3,00	3	3	3,00	3
100	20%	5	5	5,29	6	5	5,00	5	5	5,00	5
100	30%	8	7	7,64	8	7	7,00	7	7	7,00	7
100	40%	10	9	9,86	10	9	9,00	9	9	9,00	9
100	50%	13	12	12,00	12	11	11,71	12	11	11,07	12
100	60%	16	14	14,79	15	14	14,00	14	14	14,00	14
100	70%	19	17	17,93	19	16	16,93	17	16	16,93	17
100	80%	24	20	21,71	23	20	20,71	21	20	20,71	21
100	90%	30	27	27,86	29	27	27,64	29	27	27,64	29
150	10%	4	4	4,57	5	4	4,00	4	4	4,00	4
150	20%	8	7	7,57	8	7	7,00	7	7	7,00	7
150	30%	11	10	10,50	11	10	10,00	10	10	10,00	10
150	40%	14	13	13,64	14	13	13,00	13	13	13,00	13
150	50%	18	17	17,14	18	15	16,14	17	15	15,93	16
150	60%	22	21	21,00	21	19	19,79	20	19	19,29	20
150	70%	27	24	25,00	26	24	24,14	25	24	24,00	24
150	80%	34	30	31,14	32	29	29,86	31	29	29,79	30
150	90%	47	39	39,79	40	38	39,00	40	38	39,00	40

Fortsetzung auf der nächsten Seite

Fortsetzung von Tabelle A.2

			MaxIter = 10			MaxIter = 200			MaxIter = 1000		
#Blöcke	Dichte	DS	min	av	max	min	av	max	min	av	max
200	10%	5	5	5,93	6	5	5,14	6	5	5,00	5
200	20%	10	9	9,57	10	9	9,00	9	9	9,00	9
200	30%	14	13	13,43	14	12	12,93	13	12	12,79	13
200	40%	19	17	17,43	18	16	16,57	17	16	16,29	17
200	50%	23	21	21,93	22	20	20,86	21	20	20,21	21
200	60%	29	26	26,71	27	25	25,50	26	25	25,07	26
200	70%	35	32	32,57	34	30	30,93	32	30	30,50	31
200	80%	46	39	40,07	41	38	38,64	39	38	38,64	39
200	90%	60	50	51,71	54	49	50,14	52	49	50,14	52
250	10%	7	7	7,00	7	6	6,07	7	6	6,00	6
250	20%	11	11	11,43	12	11	11,00	11	10	10,93	11
250	30%	16	16	16,00	16	15	15,14	16	15	15,00	15
250	40%	23	20	20,79	21	20	20,14	21	20	20,00	20
250	50%	28	26	26,43	27	25	25,43	26	24	24,93	25
250	60%	35	31	31,79	32	30	30,93	31	30	30,43	31
250	70%	43	39	39,29	40	37	37,64	38	37	37,14	38
250	80%	54	47	48,64	50	46	47,14	48	46	47,14	48
250	90%	72	61	63,36	65	61	61,64	62	61	61,64	62
300	10%	8	7	7,93	8	7	7,07	8	7	7,00	7
300	20%	14	13	13,07	14	12	12,86	13	12	12,43	13
300	30%	19	18	18,64	19	18	18,00	18	17	17,86	18
300	40%	25	24	24,29	25	23	23,79	24	23	23,21	24
300	50%	33	30	30,79	31	29	29,71	30	28	28,93	30
300	60%	41	37	37,64	38	35	36,43	37	35	35,93	36
300	70%	53	45	46,21	47	44	44,50	46	43	43,93	45
300	80%	63	56	57,36	59	54	55,07	56	54	54,93	56
300	90%	84	71	74,57	76	71	72,86	74	71	72,86	74
400	10%	9	9	9,93	10	9	9,00	9	9	9,00	9
400	20%	17	16	16,86	17	16	16,00	16	16	16,00	16
400	30%	24	23	23,71	24	23	23,00	23	22	22,79	23
400	40%	33	30	31,21	32	30	30,36	31	29	29,71	30
400	50%	42	38	39,29	40	37	37,71	38	37	37,43	38
400	60%	51	47	48,57	50	46	46,79	47	46	46,29	47
400	70%	63	59	59,57	60	56	57,71	59	56	56,50	58
400	80%	81	71	73,43	75	70	71,14	72	70	70,14	71
400	90%	105	95	97,00	99	94	95,29	96	94	95,29	96

Tabelle A.3: Laufzeiten der lokalen Verfeinerung bei MaxIter $= 10, 200, 1000$, gerundet auf volle Sekunden. 0-Werte bedeuten also Laufzeiten unterhalb einer halben Sekunde. Wir geben wieder jeweils die minimale, durchschnittliche und maximale Laufzeit an.

		MaxIter = 10			MaxIter = 200			MaxIter = 1000		
#Blöcke	Dichte	min	av	max	min	av	max	min	av	max
50	10%	0	0	0	0	0	1	1	1	2
50	20%	0	0	1	0	0	1	1	2	3
50	30%	0	0	0	0	1	1	2	3	4
50	40%	0	0	0	0	1	1	4	4	5
50	50%	0	0	1	1	2	2	5	6	7
50	60%	0	0	1	1	1	2	6	7	9
50	70%	0	0	1	1	2	2	7	8	8
50	80%	0	0	1	1	2	2	8	9	9
50	90%	0	0	1	1	2	2	9	10	10
100	10%	0	0	1	0	1	1	3	4	4
100	20%	0	0	1	1	2	2	7	7	8
100	30%	0	0	1	3	4	4	15	16	18
100	40%	0	0	1	6	8	11	32	35	38
100	50%	0	1	1	8	11	19	44	78	108
100	60%	0	1	1	14	17	20	67	75	82
100	70%	0	1	2	19	22	31	93	99	107
100	80%	1	1	2	22	26	40	107	115	128
100	90%	1	2	3	25	26	28	121	124	129
150	10%	0	0	1	1	2	2	8	8	9
150	20%	0	0	1	6	7	8	32	35	36
150	30%	0	1	1	12	14	16	63	70	76
150	40%	0	1	2	19	21	26	99	103	107
150	50%	1	2	2	24	39	53	147	161	194
150	60%	1	2	2	34	45	69	170	229	307
150	70%	2	3	4	40	55	75	218	239	292
150	80%	3	3	5	49	62	89	245	265	300
150	90%	3	5	8	56	62	67	286	293	307

Fortsetzung auf der nächsten Seite

Fortsetzung von Tabelle A.3

		MaxIter = 10			MaxIter = 200			MaxIter = 1000		
#Blöcke	Dichte	min	av	max	min	av	max	min	av	max
200	10%	0	0	1	3	6	8	27	30	40
200	20%	0	1	1	14	15	19	70	72	75
200	30%	1	1	2	24	27	53	123	148	258
200	40%	1	2	3	34	46	73	175	230	350
200	50%	2	3	4	49	59	100	238	339	519
200	60%	3	4	6	61	79	119	302	386	597
200	70%	4	5	8	72	113	194	396	502	765
200	80%	5	7	10	89	116	178	439	474	542
200	90%	6	8	12	101	130	174	494	526	572
250	10%	0	0	1	6	12	16	52	55	61
250	20%	0	1	2	22	23	25	111	125	245
250	30%	2	2	3	36	54	74	219	235	256
250	40%	2	4	5	50	71	101	289	311	347
250	50%	3	5	7	70	93	150	404	474	599
250	60%	5	7	13	97	124	159	481	644	888
250	70%	6	8	13	121	179	272	586	791	1103
250	80%	7	12	19	149	210	264	703	769	822
250	90%	9	12	20	163	192	243	776	810	859
300	10%	0	1	1	10	18	23	79	83	99
300	20%	1	2	3	32	37	63	158	233	346
300	30%	2	3	5	57	62	68	278	305	472
300	40%	3	5	7	77	91	165	384	531	770
300	50%	5	6	10	110	126	205	531	776	1208
300	60%	7	10	15	133	187	310	702	866	1120
300	70%	8	12	17	162	249	379	839	1108	1637
300	80%	10	15	22	213	284	440	997	1121	1726
300	90%	12	20	26	231	279	403	1120	1174	1305
400	10%	0	1	2	28	34	44	144	148	155
400	20%	2	3	6	64	76	90	318	330	345
400	30%	4	6	9	103	116	150	506	583	852
400	40%	6	9	14	135	175	274	750	987	1502
400	50%	10	13	22	204	300	486	970	1267	2029
400	60%	12	16	24	259	350	498	1204	1561	2089
400	70%	16	21	31	309	454	700	1528	2477	3758
400	80%	20	37	72	371	602	840	1803	2743	3476
400	90%	23	33	45	399	472	628	1971	2054	2219

Tabelle A.4: Ergebnisse der rekursiven Verfeinerung bei MaxIter $= 10, 50, 100$. Gegeben sind wie in Tabelle A.2 die Anzahl Blöcke, die Dichte, die Farbzahl der besten Startlösung (DSatur-Adaption) sowie jeweils die minimale, durchschnittliche und maximale Farbzahl der Lösungen

			MaxIter = 10			MaxIter = 50			MaxIter = 100		
#Blöcke	Dichte	DS	min	av	max	min	av	max	min	av	max
50	1	2	2	2,07	3	2	2,07	3	2	2,07	3
50	2	3	3	3,00	3	3	3,00	3	3	3,00	3
50	3	4	5	4,00	4	4	4,00	4	4	4,00	4
50	4	5	6	5,57	6	4	5,07	6	4	5,07	6
50	5	6	7	6,64	7	6	6,14	7	6	6,00	6
50	6	7	9	7,71	8	7	7,50	8	7	7,36	8
50	7	9	11	9,50	10	9	9,00	9	9	9,00	9
50	8	11	14	11,50	12	11	11,36	12	11	11,36	12
50	9	14	17	15,07	16	14	15,07	16	14	15,07	16
100	1	3	3	3,00	3	3	3,00	3	3	3,00	3
100	2	5	5	5,21	6	5	5,00	5	5	5,00	5
100	3	7	8	7,36	8	6	6,93	7	6	6,79	7
100	4	9	10	9,50	10	9	9,00	9	8	8,93	9
100	5	11	13	11,79	12	11	11,07	12	10	10,93	11
100	6	13	16	14,21	15	13	13,64	14	13	13,29	14
100	7	17	19	17,14	18	16	16,36	17	16	16,21	17
100	8	20	24	20,86	22	19	20,14	21	19	20,14	21
100	9	26	30	27,00	28	26	26,86	28	26	26,86	28
150	1	4	4	4,36	5	4	4,00	4	4	4,00	4
150	2	7	8	7,14	8	7	7,00	7	6	6,93	7
150	3	9	11	9,93	10	9	9,71	10	9	9,29	10
150	4	13	14	13,36	14	12	12,64	13	12	12,29	13
150	5	16	18	16,57	17	16	16,00	16	15	15,86	16
150	6	20	22	20,36	21	18	19,29	20	18	19,07	20
150	7	24	27	24,21	25	23	23,71	24	23	23,21	24
150	8	29	34	29,71	31	28	28,86	30	28	28,71	30
150	9	38	47	38,64	39	37	38,21	39	37	38,21	39

Fortsetzung auf der nächsten Seite

Fortsetzung von Tabelle A.2

			MaxIter = 10			MaxIter = 50			MaxIter = 100		
#Blöcke	Dichte	DS	min	av	max	min	av	max	min	av	max
200	1	5	5	5,50	6	5	5,14	6	5	5,07	6
200	2	9	10	9,00	9	9	9,00	9	7	8,57	9
200	3	12	14	12,86	13	11	12,07	13	11	11,71	12
200	4	16	19	16,79	17	16	16,21	17	15	15,79	16
200	5	21	23	21,14	22	19	20,43	21	19	20,07	21
200	6	25	29	25,79	27	24	24,93	25	24	24,64	25
200	7	30	35	31,36	32	29	30,21	31	29	29,57	30
200	8	37	46	37,86	39	36	37,00	38	36	36,86	38
200	9	48	60	49,93	51	48	49,71	50	48	49,71	50
250	1	6	7	6,43	7	6	6,14	7	6	6,00	6
250	2	10	11	10,93	11	10	10,50	11	10	10,29	11
250	3	15	16	15,43	16	15	15,07	16	14	14,79	15
250	4	20	23	20,21	21	19	19,86	20	19	19,64	20
250	5	25	28	25,79	26	24	24,79	25	24	24,64	25
250	6	30	35	31,21	32	29	30,14	31	29	29,86	31
250	7	36	43	38,00	39	36	36,57	37	36	36,21	37
250	8	45	54	46,50	48	44	45,43	46	44	45,36	46
250	9	59	72	60,93	62	58	59,57	62	58	59,57	62

Tabelle A.5: Laufzeiten der rekursiven Verfeinerung bei MaxIter $= 10, 50, 100$, gerundet auf volle Sekunden. Wir geben wieder jeweils die minimale, durchschnittliche und maximale Laufzeit an.

		MaxIter = 10			MaxIter = 200			MaxIter = 1000		
#Blöcke	Dichte	min	av	max	min	av	max	min	av	max
50	1	0	0	0	0	0	0	0	0	0
50	2	0	0	0	0	0	1	0	0	1
50	3	0	0	1	0	0	1	0	1	1
50	4	0	0	1	0	1	1	1	2	2
50	5	0	0	1	1	2	3	2	3	5
50	6	0	1	1	2	3	4	4	7	10
50	7	1	1	2	5	6	10	10	12	15
50	8	2	2	3	8	9	12	14	18	21
50	9	2	3	5	11	14	16	22	27	31
100	1	0	0	1	0	0	1	0	0	1
100	2	0	0	1	1	1	2	2	3	3
100	3	1	2	2	5	7	11	11	13	17
100	4	3	4	6	18	21	26	36	40	54
100	5	7	10	17	40	54	76	79	104	181
100	6	14	20	36	73	89	138	144	194	270
100	7	22	31	42	114	167	237	225	303	383
100	8	34	55	94	179	256	389	331	425	524
100	9	64	87	146	291	324	430	558	610	725
150	1	0	0	1	0	1	2	1	2	2
150	2	3	4	5	11	12	17	20	23	34
150	3	11	14	21	51	62	88	84	130	172
150	4	22	32	41	139	187	287	275	369	497
150	5	49	63	82	281	338	425	544	683	1059
150	6	78	122	184	452	709	1017	902	1294	1665
150	7	143	208	301	735	942	1341	1493	2229	3398
150	8	236	342	537	1138	1689	3058	2251	3043	4129
150	9	395	534	652	1777	2182	2989	3413	3948	4637

Fortsetzung auf der nächsten Seite

Fortsetzung von Tabelle A.3										
		MaxIter = 10			MaxIter = 200			MaxIter = 1000		
#Blöcke	Dichte	min	av	max	min	av	max	min	av	max
200	1	1	1	2	3	5	10	7	10	14
200	2	10	13	18	52	55	59	100	118	167
200	3	47	56	91	226	322	454	441	598	798
200	4	110	148	213	573	728	909	1178	1589	2342
200	5	201	259	319	1080	1409	2137	2129	3055	4214
200	6	305	509	947	1949	2460	3525	3749	5052	7607
200	7	510	819	1603	2937	4311	5714	6864	9528	15042
200	8	959	1407	1971	4164	6068	11421	8598	10893	15358
200	9	1466	2077	2641	6214	7053	9317	11902	12849	15154
250	1	3	4	5	12	15	27	23	28	41
250	2	35	41	57	162	208	280	290	391	508
250	3	124	173	242	620	825	1082	1365	1739	2742
250	4	297	426	568	1745	2092	3604	3358	4244	6117
250	5	553	701	973	3177	4342	6426	6570	8020	12961
250	6	934	1408	2550	5689	8082	12763	11151	14877	20242
250	7	1624	2337	4143	9118	12877	18367	15851	23829	34680
250	8	2547	3885	5842	12756	16869	21195	23572	28571	37568
250	9	3165	5473	9933	16030	21398	37431	28633	35569	52107

Literaturverzeichnis

[1] Daniel Brelaz: New Methodes to Color the Vertices of a Graph. *Communications of the ACM*, volume 22(4), April 1978.

[2] Stephen A. Cook: The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pp. 151–158. 1971.

[3] Thiago F. de Noronha and Celso C. Ribeiro: Routing and wavelength assignment by partition coloring. *European Journal of Operational Research*, 2004. Noch nicht veröffentlicht. `http://dx.doi.org/10.1016/j.ejor.2004.09.007`.

[4] Naveen Garg, Goran Konjevod and R. Ravi: A polylogarithmic approximation algorithm for the group Steiner tree problem. *Journal of Algorithms*, volume 37(1):pp. 66–84, 2000.

[5] Martin Charles Golumbic: *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1978.

[6] Lars Kaderali, Alina Deshpande, John P. Nolan and P. Scott White: Primerdesign for multiplexed genotyping. *Nucleid Acid Research*, volume 31(6), 2003.

[7] R. M. Karp: Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher (Editors), *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York, 1972.

[8] D. König: Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Mathematische Annalen*, volume 77:pp. 453–465, 1916.

[9] Bernhard Korte and Jens Vygen: *Combinatorial Optimization: Theory and Algorithms*. Springer, 2nd edition, 2002.

[10] Guangzhi Li and Rahul Simha: The Partition Coloring Problem and its Application to Wavelength Routing and Assignment. In *Proceedings of the First Workshop on Optical Networks*. 2000.

[11] D. Matula, G. Marble and J. Isaacson: Graph coloring algorithms. In R. C. Read (Editor), *Graph Theory and Computing*, pp. 109–122. Academic Press, 1972.

[12] Michal Penn and Stas Rozenfeld: Approximation Algorithm for the Group Steiner Network Problem, October 2003.
`http://iew3.technion.ac.il/Home/Users/mpenn.html`.

[13] G. Reich and P. Widmayer: Beyond Steiner's Problem: a VLSI oriented generalization. *Lecture Notes in Computer Science*, volume 711:pp. 196–211, 1989.

[14] Thomas J. Schaefer: The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pp. 216–226. ACM Press, New York, 1978.

[15] D. Welsh and M. Powell: An upper bound to the chromatic number of a graph and its application to time-table problems. *The Computer Journal*, volume 10:pp. 85–86, 1967.